P-SEMP: A PLATFORM FOR SYSTEMS ENGINEERING MODELING AND PLANNING

A Thesis
Presented to
The Academic Faculty

By

Kevin A. Reilley

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology

May 2021

P-SEMP: A PLATFORM FOR SYSTEMS ENGINEERING MODELING AND PLANNING

Approved by:

Dr. Dimitri N. Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Daniel P. Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Eric M. Feron
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Alicia M. Sudol
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Russell S. Peak
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Bjorn F. Cole
External
Lockheed Martin

Date Approved: December 19, 2020

ACKNOWLEDGEMENTS

The thesis is an endurance race. There are ups and downs, times of great productivity, and times of despair. It is impossible to finish this race without the support of others, and I would like to acknowledge some of them here.

I must first thank the greatest supporter of my odyssey in graduate school. My grandfather Robert H. Reilley always encouraged the process, always cheered me on, and always wished the best for me. After my internship at JPL in 2015, he would wear a JPL hat around and tell his friends how his grandson was getting a PhD at Georgia Tech, and had worked with NASA. Even when the process was at its most difficult, he encouraged me to press on, never to drop out, and to finish the job. There is no end to the remorse I have that this race should have endured too long for him to witness the finish.

Additionally, I must recognize my committee, and first of all my advisor Dr. Dimitri Mavris. Dr. Mavris gave me the opportunity to join ASDL when all my other options had gone away, and together with committee member Dr. Russell S. Peak helped to find and provide funding for me as the years went by, without which this effort would have been impossible. The guidance on the argument from Dr. Mavris, and the tutelage of Dr. Peak, enabled me to pursue this thesis. Additionally, I must thank Dr. Alicia Sudol, Dr. Bjorn Cole, Dr. Eric M. Feron, and Dr. Daniel P. Schrage. Dr. Sudol and Dr. Cole especially spent many hours assisting me in the process of polishing various aspects of the argument of the thesis.

Discussion about issues in the formulation of the thesis among my colleagues in the Aerospace Systems Design Laboratory was crucial to understanding the process and getting the job done. Many hours of mentorship were invested by Dr. Stephen Edwards and Dr. Selcuk Cimtalay, for which I am grateful. Among my peers, many hours of discussion with Eugina Mendez-Ramos, Nicole Robertson, John Robinson,

Marlin Ballard, Raphael Gautier, Manuel Diaz, among others aided in the process.

Outside of the lab, two longtime friends provided constant support and encouragement. Farzon Lotfi and Bhanu Kumar helped tremendously in bearing the burdens of the thesis, through discussion and permitting me to share my grievances freely, and perhaps too verbosely. At one time, Farzon in particular helped make sure I had a roof over my head. Bhanu helped me navigate the wild rental situations of Pasadena. I could not have made it through this process without their support.

I must thank my parents for providing support throughout the thesis journey. They helped bear the costs of my undergraduate, and they helped support some of the costs during the odyssey through graduate school. They gave me plenty of advice on communication, though I may not have followed as much of it as they think I should.

Finally, my fiance and shortly wife Kristiana, who has seen the full range of optimism and despair in the course of this process during the four years we have been together. She has borne the brunt of my verbosity and rants about the process, and I can probably never fully make up for it. Her patience, her graciousness and forbearing of my continuing this race ahead of her, her candor, and her assistance have been there for me through the critical stretches. As my thesis ends, and our life begins, I get to pass on the baton to her so that she may take up this race herself.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Systems engineering management and planning has long been a realm dominated by arcane standards, by the weight of years of practice, and by authority. However, with technological advances and the desire to solve socio-technical problems at the level of increasingly complex systems, authority alone is no longer sufficient for the justification of systems engineering practice. As new systems engineering methodologies are bought and sold in the transition towards model-based systems engineering, there is an imperative for the systems engineering practitioner to develop new techniques for estimating project performance before project completion. That is, whether debating appropriate corrective actions for a project at risk of going over budget or over schedule, or when planning a new systems engineering methodology, the systems engineer must forecast planned performance of systems engineering tasks. To this end, the International Council of Systems Engineers (INCOSE) and others have sought to bolster systems engineering measurement and the development of standardized leading indicators of systems engineering performance, which are thought to give insight into future performance in the course of program performance. Recent efforts have produced models of systems engineering performance; however, no model is yet sufficient for addressing which tasks in support of standardized processes should be planned in a systems engineering methodology. This document lays out how such a capability might be implemented by a platform for the numerical comparison of systems engineering methodologies. The idea of a platform for systems engineering modeling and planning is called P-SEMP.

There are two threads in this document: a thesis and a methodology. First and foremost, the document is a thesis. The thesis, called at times the P-SEMP Thesis, is a formal argument as to how to address the problem of systems engineering task planning constructed on the basis of gaps, research questions, hypotheses, experiments,

and their results. The P-SEMP Thesis aims to prove the best means for determining which systems engineering methodologies, and in particular which methods for a given systems engineering process, are better or worse. Enabling the argument of the P-SEMP Thesis is the P-SEMP Methodology, which is rooted in the fundamentals of modeling and simulation theory but made specific to the class of problems involved in systems engineering methodology comparison. The P-SEMP Methodology describes how to build a platform for P-SEMP and what a platform may entail, and the methodology is supported by a conceptual architecture description. The combined product of the P-SEMP Methodology and conceptual architecture description is a recipe: first, a recipe in terms of the proposed experiments, and then a recipe for the experimental results and conclusions of the P-SEMP Thesis and how its findings may be further applied.

In order to render the P-SEMP Thesis manageable in scope, the focus will be placed on tasks surrounding the systems engineering process of validation. Validation, in different senses, can occur both early and late in the system life cycle. While validation is a controversial term, many authors agree that efforts around feasibility assessment, requirements quantification, and the early evaluation of system architectures and design against these requirements are crucial steps in early-phase validation to ensure that the system will meet stakeholder expectations before proceeding with the entirety of the system lifecycle. Concretely, as proposed sets of tasks, a portion of an Object-Oriented Systems Engineering Methodology-inspired process for Spacecraft Requirements Derivation is compared against the State Analysis Model Development method, and subsequently a third method is proposed as well regarding validation concepts. These methods for validation will be modeled and compared using the tooling developed in support of the argument for a platform for systems engineering modeling and planning, the P-SEMP Thesis, and be constructed according to the P-SEMP Methodology with results as shown in the conceptual architecture description

for Platform 0.1 and Platform 1.0. The result of the experimental efforts culminates, in a concrete sense, with a domain-specific language for describing tasks in a manner suitable for simulation of the method models. However, leading indicator models are not forgotten; one in particular is replicated and added to a system modeling environment alongside the method models — however, serious issues in parameterization are uncovered in these leading indicator models and they may not provide much insight towards task planning. Due to these issues and more, a hybrid model proved infeasible in the current situation, leading to the evolution from conceptual Platform 0.1 to the final Platform 1.0.

Additionally, as the Spacecraft Requirements Derivation method is proposed specifically for a canonical system FireSat, specific modeling practice in SysML will be proposed to represent the third proposed SE methodology being compared, which requires representation of designs of experiments and probability distributions in the course of ensuring system feasibility. Another motivation for incorporating these expressions into a system model is to ensure the correctness of analytical models which underlay validation processes. This correctness is established by model verification and validation. As these analytical models represent the system from different perspectives, it is beneficial for them to be closely coupled to a unified system model depiction. However, a gap exists where while such capability is known for Multidisciplinary Design Optimization (MDO) and system models, it does not yet exist for Robust Design Simulation (RDS) or techniques for probabilistic or uncertain design processes in conjunction with a system model. Such a technology helps to support the activities above and improves confidence in the results of the early-phase system validation actions.

In summation, according to the argument of the P-SEMP Thesis and the practice of the P-SEMP Methodology, a leading indicator model is replicated and found wanting. Systems engineering method model simulations are formulated, and a domain-

specific language is created to capture them in the system model for exploration of task architecture. Finally, broader description of designs of experiments and probability are incorporated to improve analytical integration capabilities required for full validation activities in support of greater systems engineering methodology capability. Synthesizing the experimental results is the P-SEMP conceptual architecture Platform 1.0, which serves as a new baseline for systems engineering task planning and comparison, and which places the results into the greater context of how to build a platform and use the platform. Altogether, these pieces outline a platform for systems engineering modeling and planning on the basis of constructing a suitable platform through various models and exercising the resulting platform, thus improving systems engineering methodology analysis. Specifically, the thesis demonstrates how P-SEMP is the first known technique for SE methodology selection that supports 1) mathematical models of task performance, 2) analysis of SE methods as tasks in a concrete sense, 3) inclusion likewise of soft or subjective criteria, and 4) expandability to investigate new or different SE method proposals in a unified and effective manner.

1
INTRODUCTION AND MOTIVATION

> *"If we take in our hand any volume; of divinity or school metaphysics, for instance; let us ask, Does it contain any abstract reasoning concerning quantity or number? No. Does it contain any experimental reasoning concerning matter of fact and existence? No. Commit it then to the flames: for it can contain nothing but sophistry and illusion"*
>
> – David Hume, *An Enquiry Concerning Human Understanding*

The purpose of this thesis is to provide insight on how the cost and time to bring systems into being might be reduced through quantitative analytical studies. By accelerating system design and development and providing measurable improvement, businesses can exploit markets more efficiently and governments can accomplish their objectives more effectively. Systems design and development, however, is a vast field and often treated in generic terms. Throughout this thesis, systems development issues will be discussed at increasing levels of specificity, and objective will be to provide contributions reducing cost and lead-time in a few focused areas. First and foremost, this entails an understanding of what it means to apply a methodology to the practice of Systems Engineering (SE).

There are a few principle challenges in determining how to perform SE which arise due to its frequent application as a business process in military and commercial sectors — that is, information about SE is often obscured from public view. The obscurity of data means that a problem of sparsity exists, so that it may be difficult to use machine learning and data science to improve the practice of SE outside the sphere of an organization or program. In part due to the lack of visibility, there are often confounding factors in the presentation of SE data. For example, it may be difficult

to know whether issues in SE arise from product complexity or from organizational mismanagement. Causal relationships to trends in SE metrics may not be possible to establish on the basis of measurement and experiment alone. Compounding this difficulty is a confusion of the terms tool, method, and process[1]. These issues with understanding SE through data are important because this data is often the basis for decision-making on SE. Fortunately, the practice of SE has been shown to have a positive impact — up to a limit — on the organizational bottom-line[2]. However, even the existing studies which have collected survey data such as Honour [2] focus on the impact of SE technical processes, thus providing an indication of whether SE practice has a positive impact, and not whether any particular methodology of SE is advantageous.

1.1 Towards a Definition of Systems Engineering Methodology

Several projects have established various means for evaluating the costs of SE. Engineering endeavors take much interest in how to perform SE, as SE is a key factor towards ensuring delivered systems conform to specification. While some of these studies are more concerned towards how much SE ought to be performed (e.g. Figure 1.1), the interest here will be to what extent the existing models and data inform how best to practice SE.

SE is defined in the INCOSE Handbook as related to the terms "interdisciplinary, iterative, sociotechnical, and wholeness"[3]. The handbook says that that SE may be considered "a perspective, a process, and a profession"[3]. SE as such seeks to manage complexity and change, especially in the design of new systems or revisions to existing complex systems. The reason for this, explained by both the INCOSE Handbook[3] and Martin[4], is that the majority of a system's life-cycle cost (LCC) will be locked-in during early phases of development; the INCOSE Handbook states "that when 20% of the actual cost has been accrued, 80% of the total LCC has already

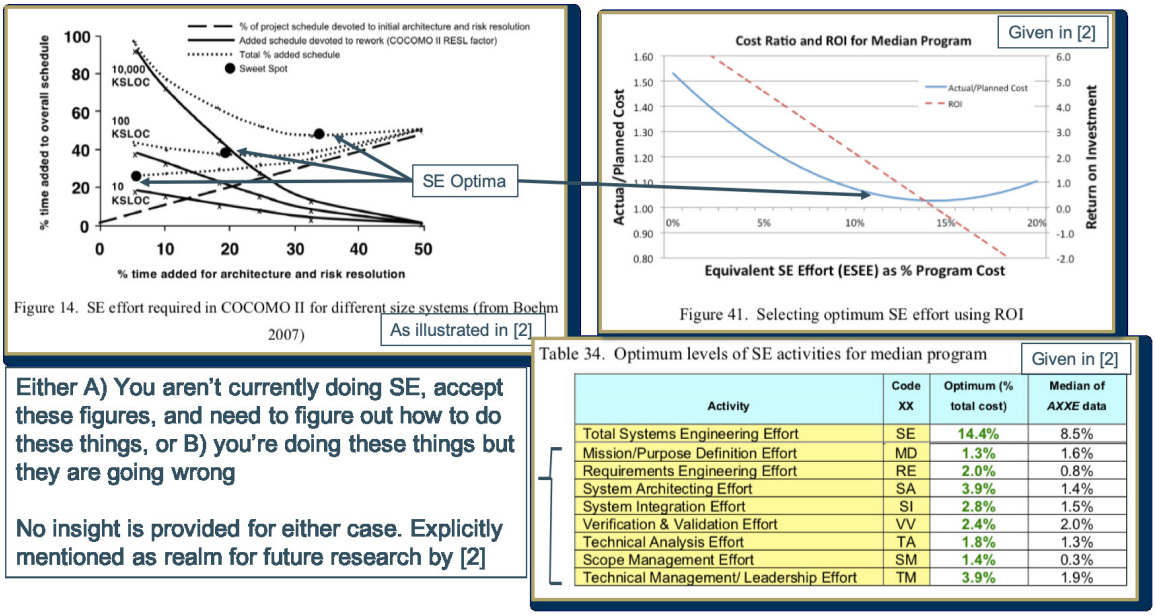


Figure 1.1: Combination of portrayals from Honour[2] which illustrate the historically optimal application of SE effort

been committed"[3]. Compounding this effect, a significant amount of effort must be applied to SE itself: the SE-ROI study has even found an historical optimum of about 14% total effort to be applied to SE to minimum cost overrun, schedule overrun, and maximize program success[2], illustrated in Figure 1.1. Any decisions made early in the development process should be as correct as possible to prevent costly rework, redesign, and overruns later in the system life cycle. Madni and Purohit[5] recently published a design-structure matrix representation of the recommended processes for SE according to ISO 15288[6] and INCOSE[3], and this figure is given in Figure 1.4 for illustrative purposes.

Systems Engineering Methodology is defined "as the application of related processes, methods, and tools to a class of problems that all have something in common"[1]. Usually Estefan, Martin, and others present the relationship of the elements of methodology as in Figure 1.2. According to Estefan[1] and Martin[4], process describes what will be done, methods describe how to accomplish the tasks in the process, and tools enhance the ability of engineers to accomplish these tasks. Further-

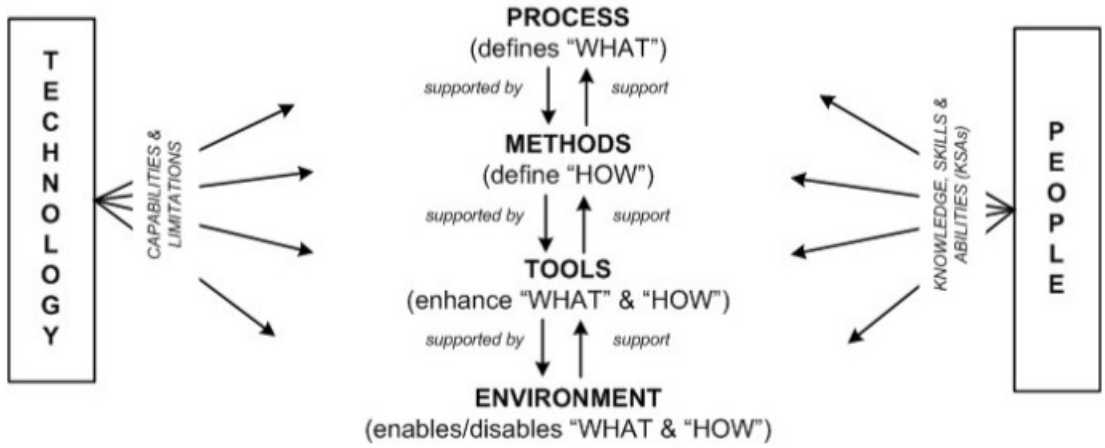


Figure 1.2: The PMTE Elements of Systems Engineering Methodology according to Estefan[1] and Martin[4]

more, all of this material is performed within an environment that enables or disables work towards the tasks; Martin[4] defines this Process-Methods-Tools-Environment framework as a PMTE paradigm for understanding and balancing the aspects of Systems Engineering practice. First however, further definition of the SE processes is necessary.

1.1.1 Definition of Systems Engineering Processes

The SE process is defined according to standards, and the application of these standards is a major role of systems engineers[7]. The life cycle processes are fairly well-known, defined in multiple sources like EIA 632, CMMI, and ISO 15288[7]; the ISO 15288 processes are shown in Figure 1.3 from [7] and [6].

Factors influencing processes include input, output, controls, and enablers[8]. Madni and Purohit recently translated all of this interface and process information into a design structure matrix, and the graphic from their paper is given in Figure 1.4.

Overall, the key issue is that these processes occur at different times and to different extents throughout the system life cycle. The system life cycle according to the

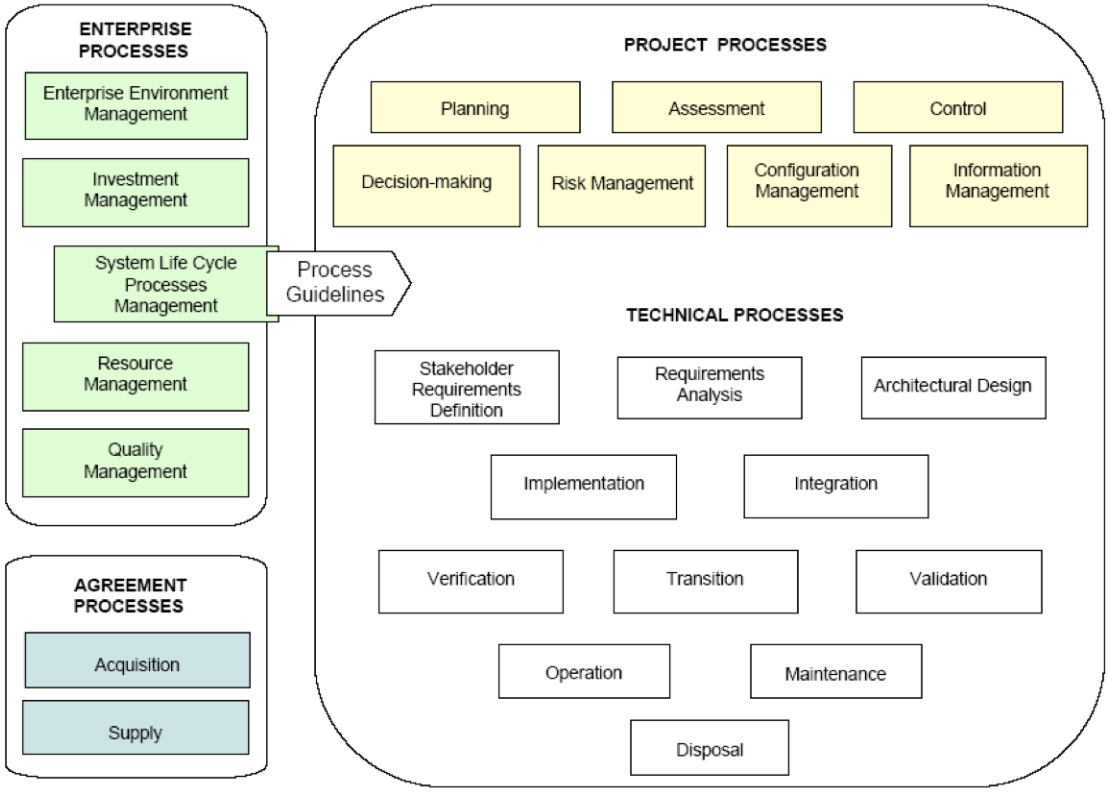


Figure 1.3: Standard processes from [7] citing [6]

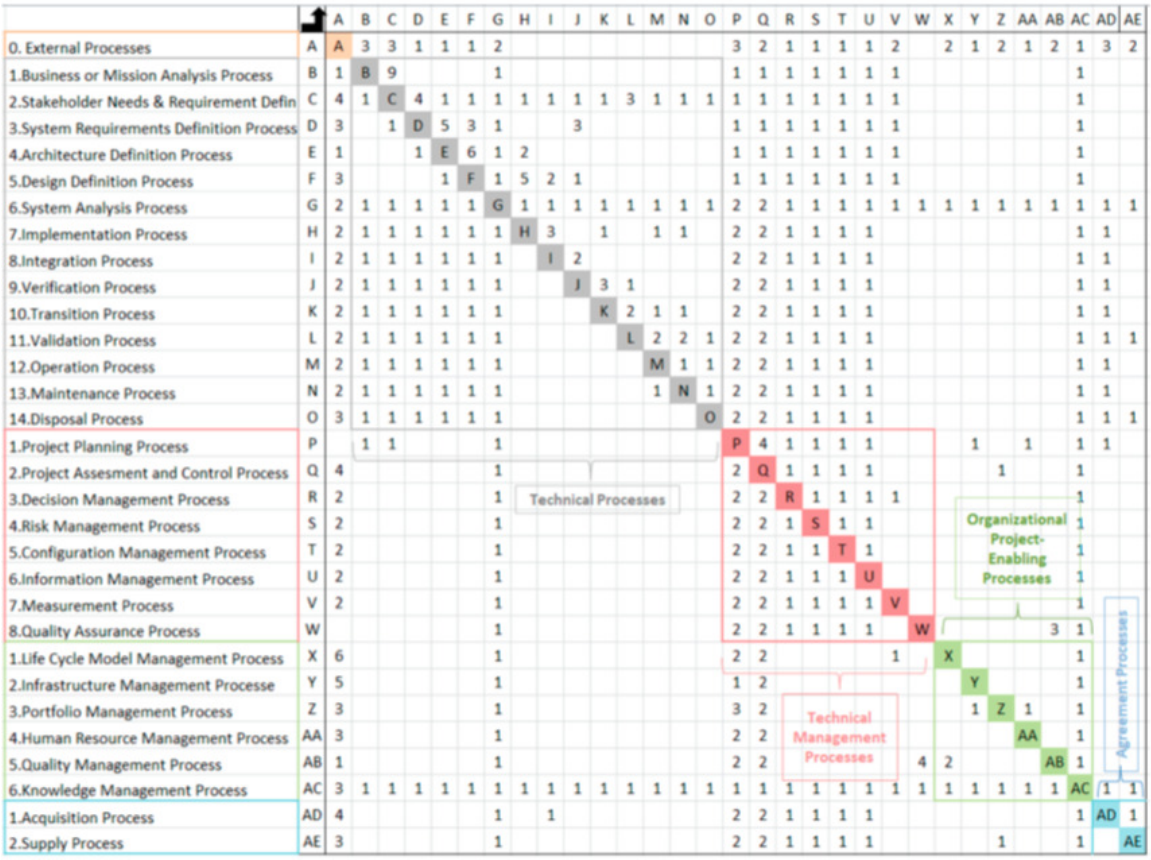


Figure 1.4: Processes in Systems Engineering according to Madni and Purohit[5]

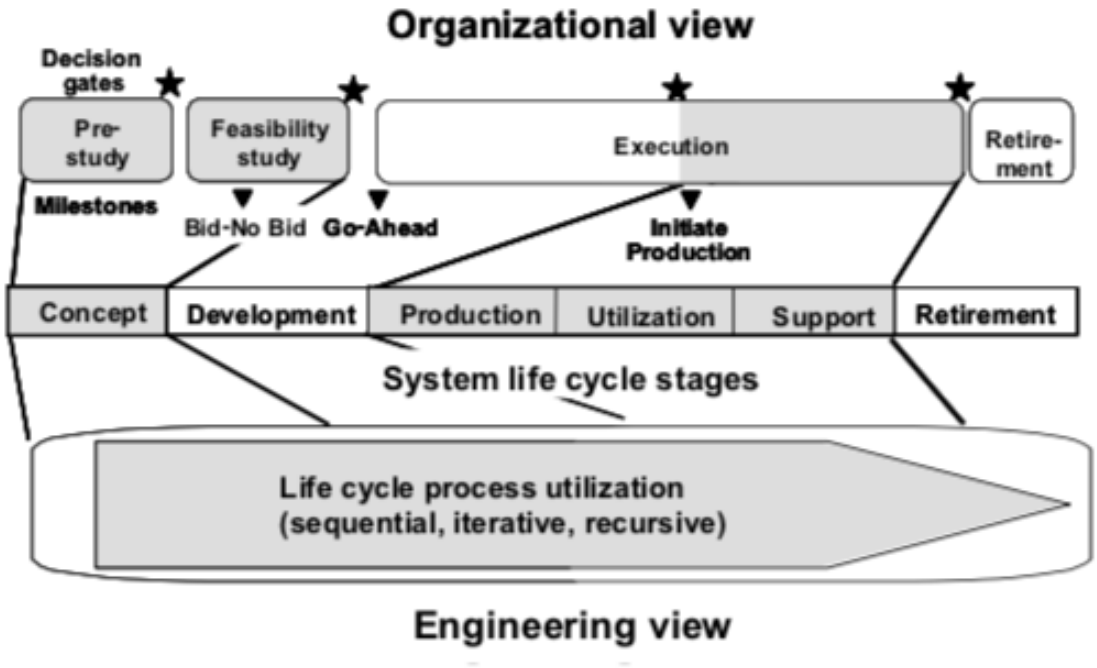


Figure 1.5: System Life Cycle according to [8]

ISO standards (15288 and guides) is shown in Figure 1.5 from [8].

In order to clarify the differences in terminology across standards for common comparison, a common SE ontology has been put forward[2, 9][10]. This common ontology collapses the life cycle down to the shared processes across multiple standards, but has not been updated since the mid-2000s. That said, this literature forms the basis for understanding *what* is done for SE. The remaining, and very important question, is *how* to do this work of the life cycle processes.

1.1.2 Methodology for Traditional SE

Systems Engineering is a tradition of the 20th century. A primary SE methodology which arose during the past 80 years according to Grady[11], other than ad-hoc approaches, is known as Structured Analysis. While Martin[4] does not name the methodology as such, his descriptions shares many similarities with Grady[11]. Structured Analysis relies on a functional analysis and allocation process to deter-

Process	Methods	Tools	Environment
Requirements Analysis	Initial System Analysis & Design Concept Synthesis	• System definition document, • requirements allocation sheets, • functional flow diagram, • data flow diagram, • behavioral schematic block diagram, • physical architecture block diagram • More… (Grady2006 p. 107, fig 3.3-1, defines all traditional structured analysis work products)	Editor software/equipment appropriate for the work products (could be ms office + visio, or typewriter + drafting board)
	Functional Analysis		
	Perfo Reqs Analysis		
Architectural Design	Architecture Synthesis		
	Specialty engineering		
	Timing & Environmental Analysis		
	Detailed Specification Analyses & Documentation Management		

Figure 1.6: PMTE for Structured Analysis as described in Grady[11]

mine what the system must do for the customer, and then a requirements derivation process to determine the extent to which the system must perform. The functional architecture and the derived performance requirements are then allocated to the system architecture, which is some sort of physical decomposition, often in block-diagram format. Throughout, there are a series of products (diagrams, tables, reports) which are produced to describe how the system operates and what it is. The products of SE methodologies such as structured analysis serve a purpose towards clarifying whether the system design does what is intended. The intent of a system is to meet some customer need; detailed documentation of how the system relates to the customer need is provided through the various *work products*. Authors will divide these *work products* into various documents and reports corresponding to the system life cycle and certain reviews along the way. Martin[4] gives specification documents, system design documents, interface requirement specifications, manuals, and other kinds of documents as the outputs of the process. Blanchard and Fabrycky[12] also list many documents and reports and levels of specification documentation which are produced along the life cycle. The significance of all this documentation work is to make explicit how the *system architecture* will correspond to the requirements, especially with regards to any *emergent behavior*. The architecture of a system is defined by ISO 42010 as the "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution"[13]. Rasmussen notes in his lecture that "a system is greater than the sum of its parts"

in that "systems are intrinsically about what is added through interaction"; that is, the system architecture is a real phenomenon which gives rise to the real-world behavior of systems according to their physical, digital, biological, etc, components[14]. In the terms of ISO 42010, the block diagrams and flow diagrams which represent the intended architecture in the SE documentation are views, viewpoints, architecture descriptions, etc., but not the system architecture itself[13]. This is why the SE methodologies include extreme emphasis on experimental verification and validation[15][4][3], because the *concept* of how the system ought to meet the requirements may not have been translated through manufacturing and/or acquisition to the real thing. All that said, it is possible to map some of Grady's structured analysis products against PMTE to better understand its recipe for SE, as shown in Figure 1.6.

1.2 The Evolution Towards Model-Based Systems Engineering

As SE is partly social in nature, its practice is strongly influenced by its history. The truth of this statement can be clearly seen with model-based SE. Since Wymore's book in 1993[16], the term Model-Based Systems Engineering (MBSE) has seen widespread use in professional societies, businesses, governments, and academic literature. While Wymore had previously established[17] many of the tenets he later renamed as MBSE, the term has been under constant evolution. The current agreed definition of MBSE is "the formalized application of modeling to support system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases"[3, INCOSE Handbook citing INCOSE Vision 2020]. In fact, INCOSE goes so far as to state that "In an MBSE approach, much of [the SE work product detail] is captured in a system model or set of models"[3], agreeing with Delligatti[18] that instead of separately authoring content in text or graphical form within documents, the Systems Engineers will author models which generate documents at a later time (i.e. during reviews).

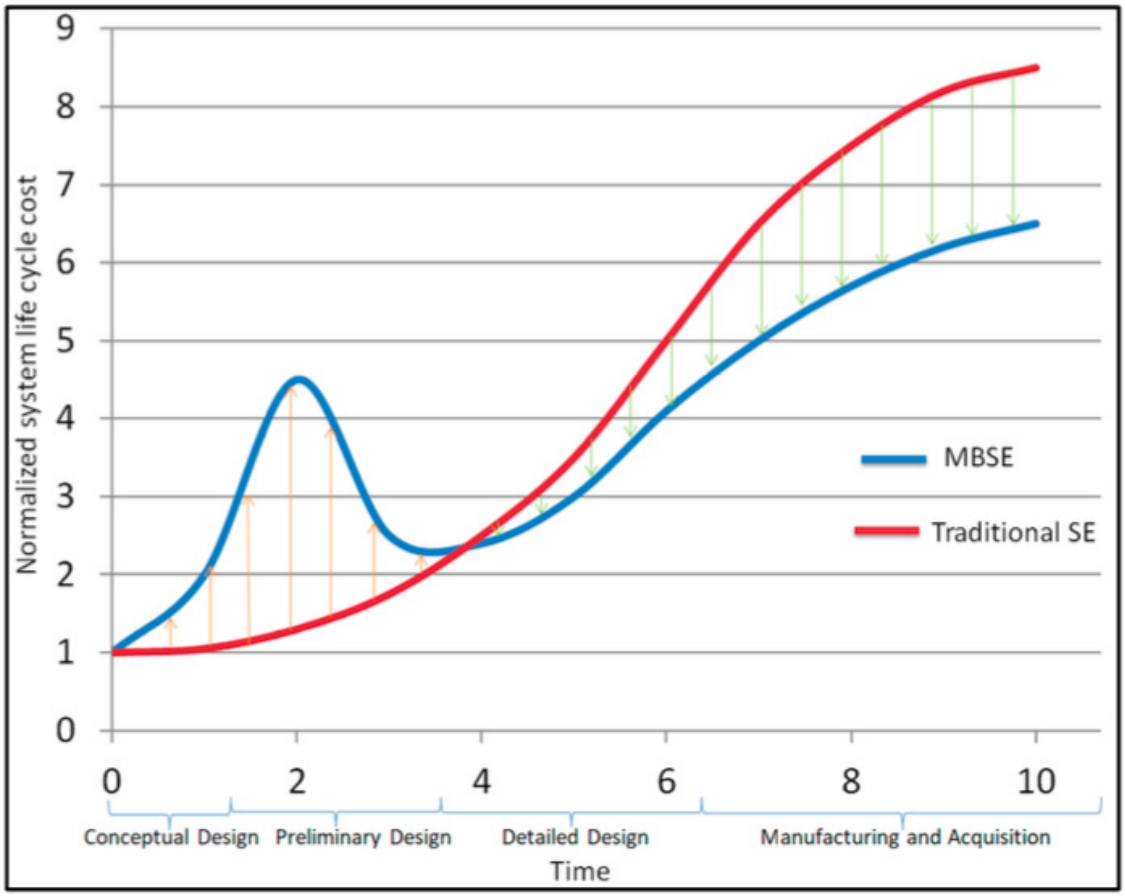


Figure 1.7: Relative Cost Profile for SE vs MBSE over System Life Cycle[5]

However, as described by Chapman, Bahill, and Wymore, the early concepts of MBSE involved the creation of seven system definition documents across the system life cycle[19]. The difference from traditional SE methodologies, however, was to be in the *content* of these documents. The seven documents were to guide the elaboration of a set-theoretical systems theory notation of the product under development. For Chapman et al, the information across these documents would be, in paraphrase of INCOSE, maintained, synchronized, and assessed for quality through correctness, completeness, and consistency[3] through mathematical proof of the system theorems and simulation of the resulting set-theoretical system model according to operational conditions or even with hardware in the loop[19]. A key point of emphasis again, is that the purpose of authoring the set-theoretical system model based on [16] in these

documents is to create real systems as shown in the latter half of [19] for pinewood derby and safe-rail control. This tension, between the authoring and validation or verification of system models and the realization of actual systems in the real world, is a key point of contention in the MBSE community with real consequences. In Figure 1.7, one consequence is illustrated by Madni and Purohit's normalized cost comparison for traditional SE vs MBSE. The key issue raised by Madni and Purohit is that without careful design of the methods for the SE processes engaged in for MBSE, the increase in modeling effort early in the life cycle may outweigh any savings later on, negating the entire purpose of moving to MBSE[5]. While the authors go further to try to indicate classes of products for which this inversion is less likely, it is also important to consider how the work is done which leads to the overspending; methodologies for MBSE must be considered.

1.2.1 Methodology for MBSE

For the field of MBSE, there exists a standard reference work which provides a "'Catalog of MBSE life cycle methodologies"'[1]. The first section of this well-known paper by Estefan presents a set of general definitions, including for SE methodology, system life cycle "models"/processes, and a mathematical overview of MBSE. While the description of life cycle models/processes and mathematical foundations can be found elsewhere, Estefan provides a clear description of what a SE methodology entails. That is, a SE methodology indicates *WHAT* must be done, *HOW* it will be done, *WHO* will do it, as well as the tools and technologies available to enable the work in the context of an environment — the product development ecosystem. For Estefan, *WHAT* must be done entails the work of SE life cycle processes, such as requirements analysis, architectural/system design, and requirements verification. The *HOW* is the method, or specific steps, for completing the processes described by the methodology. After providing these definitions and more, the middle part of the paper presents

descriptions for six popular MBSE methodologies, including:

1. IBM Harmony-SE
2. INCOSE Object-Oriented Systems Engineering Method (OOSEM)
3. IBM Rational Unified Process for Systems Engineering (RUP SE) for Model-Driven Systems Development (MDSD)
4. Vitech Model-Based System Engineering (MBSE) Methodology
5. JPL State Analysis (SA)
6. Dori Object-Process Methodology (OPM)

The final portion of Estefan's paper gives an overview of enabling technologies for MBSE, including the Object Management Group (OMG) Unified Modeling Language (UML) and Systems Modeling Language (SysML), the OMG Model-Driven Architecture (MDA), and the OMG Foundational Subset for Executable UML (fUML). While Estefan's paper presents a key resource for situating beginners in MBSE, the key takeaway here is the definition of SE methodology:

Definition 1 "A [SE] methodology can be defined as a collection of related processes, methods, and tools."[1]

Definition 1 serves as the key set of descriptors for any given SE methodology. In order to determine how to best perform SE, *the key objective of this dissertation*, it will be necessary to compare different methods for identical processes tailored to a given system and organization. The specific definition and structure of SE methodologies to be investigated will be presented later on the basis of additional sources.

While there are many works expanding on the SE methodologies described by Estefan, especially the IBM methods and OOSEM[20], another method which should be given additional detail is State Analysis (SA). Grady[11] makes a distinction in

Process	Methods	Tools	Environment
Stakeholder needs/Requirements Analysis	ID Needs – control objectives (goal – "constraint on the value history of a state variable over a time interval")	Command models (statechart/goal network/statemachine/ontology)	Powerpoint, diagraming tool (e.g. Dia), magicdraw
Architectural Design	ID state variables, plant under control, and representation	State models (statechart/goal network/statemachine/ontology)	
	Define state models for state variables	State models (statechart/goal network/statemachine/ontology)	
	ID measures to estimate state variables and define	Measurement models (statechart/goal network/statemachine/ontology)	
	Define measurement models	Measurement models (statechart/goal network/statemachine/ontology)	
	Define command models	Command models (statechart/goal network/statemachine/ontology)	
	Repeat on discovered state variables	State models (statechart/goal network/statemachine/ontology)	
Stakeholder needs/Requirements Analysis	Elaborate objectives & iterate process until mission is covered	Command models (statechart/goal network/statemachine/ontology)	

Figure 1.8: PMTE for State Analysis as described in [21, 22]

architectural terms between "form follows function," as the class of methodologies based on functional analysis, and methodologies where "both form and function are important". SA presents the latter description view, in contrast to Structured Analysis presented earlier. Under Structured Analysis architectural design is focused on the physical, electrical, and digital layout of the system and its components and interactions according to requirements and functions, state-analysis focuses the architecture around defining state variables, models of states, plants, controllers, associated measures, etc[21, 22]. It could be said that in SA, the system is much more *cyber-physical*. The PMTE elements of SA from the literature are shown in Figure 1.8.

Clearly, there are different means of accomplishing the SE life cycle processes which can result in viable products, whether using a document-based or model-based paradigm for SE. The existence of alternatives implies a need to *decide* from among the alternatives in order to perform SE — or even to support construction of new SE methodology.

1.3 Supporting Decision-Making on Systems Engineering Methodology

Understanding that a Systems Engineering methodology is composed of Process, Methods, Tools, and Environment (PMTE) and influenced by technology as well

as the Knowledge, Skills, and Abilities (KSAs) of the people involved[4], represents a first step in making a decision as to how to organize the elements needed for a Systems Engineering methodology. The next step, is that the practice of Systems Engineering is measurable via leading (indicative of future performance) and lagging (indicative of the work already completed) indicators. These two steps are key for enabling decision-making on the Systems Engineering methodology for design of the methodology, selection of a methodology, or systematic tailoring of a methodology.

1.4 Setting the Stage

While Systems Engineers may find the need to decide on a SE methodology trivial, it is worth understanding this decision-making problem from a wider perspective. Readers of this thesis may be aerospace disciplinarians — experts in aerodynamics or structures, propulsion, etc. Or perhaps the reader is a specialist in multi-disciplinary design, analysis, and optimization. Further, perhaps the reader is part of an organization which has not yet implemented SE, or is in the early stages of SE competency. How then would such a reader **determine whether any current or proposed SE activity is an improvement over any other?** SE is vast and includes many methodologies, many commercial vendors peddling "all-encompassing" solutions, and all of which is far removed from a disciplinarian's usual domain of expertise.

Ideally in this situation, there should exist a capability to enable decision-making by trading various qualities of the proposed SE activities. Such an environment may help provide clarity in the absence of extensive SE experience or expertise. Qualitative techniques alone have limitations. Quantitative techniques for comparing proposed SE approaches are preferable, as all decision-makers are at some point interested in cost. For these purposes, this dissertation will explore the existing cost models for SE, investigate whether these models are sufficient to address the problem posed above, and to formulate improvements which augment the current state-of-the-art.

1.5 The Bottom-Line: Economic Analysis

In addition to the thought experiment above, recent economic analysis has shown that different approaches to SE will alter the schedule of expenditures during a program. Most recently, Madni and Purohit[5] found that Model-Based Systems Engineering (MBSE) results in greater up-front costs than traditional Systems Engineering but lower costs later in the system life cycle. However, Madni and Purohit did not establish that MBSE saves money over traditional approaches, and acknowledges that particular implementations may cause variations in the results. Madni and Purohit also showcase a common problem in quantitative studies of SE when they write, "historical data (evidence)... was a keynote address at the 2016 No Magic World Symposium"[5], and this data point was the basis for the earlier Figure 1.7. Such a singular, un-vetted data point is not evidence in any other domain. Systems Engineers, plagued by restrictions on the publication of metrics, communicate in a world of data-sparsity. Other works attempt to build up larger sets of data regarding return-on-investment for MBSE such as Gooden 2016[23], but these seem to be unfinished. A clear need exists to provide a platform for rational, sound consideration of SE tasks in the context of a given project or program, in order to ensure that program changes (e.g. the adoption of MBSE in place of SE down to the level of specific tasks employees partake in) perform as desired.

1.6 Research Objective

The objective of this proposal is to improve the tailoring and design of SE methodology through modeling and simulation of SE processes with a basis on empiricism to optimize value for projects, programs, and organizations. In short, the result of this study should enable the construction of *predictive models* based on some description of the planned SE tasks. The ability to construct these models is, in effect, a plat-

form for combining a variety of measures and available calibration data to forecast the impact of changes in methodology, for the purpose of planning improvements to SE practice. This is exactly the idea behind a Platform for Systems Engineering Modeling and Planning (P-SEMP). Specifically, the desire is to better understand through quantitative comparison the relative merits of Systems Engineering methods for Systems Engineering processes. The following chapters will establish the existing models of SE effort, how these models fall short in the context of process improvement, useful techniques from business process re-engineering and situational method engineering, and finally an experimental plan for constructing the necessary investigative platform. Some preliminary questions will guide the exploration.

1.7 Preliminary Questions

There are a few Preliminary Research Questions (PRQ) which must be established before continuing. In accordance with the research objective, firstly the SE process must be measured. Thus PRQ-1:

PRQ 1 How should the SE process be measured?

In order to better reach the objective, two additional questions must be derived from PRQ 1. These questions seek to establish better how such a measure can enable a predictive capacity. Firstly, a set of models is required to better enable planning of the SE processes. That is, PRQ-2:

PRQ 2 How is modeling and simulation applied to the SE process?

With proper models, simulations of model parameters should give insight into the structure of the SE tasks in the context of an organization's tools and environment, such that outcomes are predicted to some degree of certainty (probabilistic model) and course-corrections can be recommended (conditional probability). These simulations should involve characteristics both of the organization, the tasks/plan, and

the system. In this way, such a prediction is contextualized by the organization performing SE. Finally, the purpose of such an exercise is to make decisions on a SE methodological implementation. This is PRQ-3:

PRQ 3 How are decisions made on the content of the SE process?

For example, any given measure (e.g. number of nodes, number of requirements, number of staff...) is only useful insofar as it can result in action by the organization. Decision-makers need a rational basis for their decisions[24], which is often argued to be solely in terms of the cost/profit measure (e.g. actual/predicted cost, actual/predicted profit). In the case where such measure is not available, nor actual/predicted schedule, nor performance/success/"goodness" of the outcome of engineering project, then the supplementary measures should have some correlation with others. That is, any measure (e.g. number of nodes, number of requirements, number of staff...) or composition of measures should provide insight into quantities better understood by decision-makers, such as cost/profit. The following chapter will seek to elaborate and possibly answer PRQ-1, -2, and -3.

2

BENCHMARKING BASELINE APPROACHES IN SYSTEMS ENGINEERING MEASUREMENT

"Wisdom comes through suffering"

– Aeschylus, *Oresteia*

Systems Engineering does not come for free. There is a price to the activities involved, and this price has been learned over many years of SE practice. Therefore, organizations engaged in SE practice have a motivation to ensure that 1) they are aware of the cost, and 2) that their SE implementation is optimally effective. Accordingly, there is substantial literature in modeling the effort required to practice SE. Models of this sort largely rely on the accumulation of historical data and expert knowledge, with limited modeling and simulation. These studies exist for two purposes: 1) to determine, for planning purposes, the likely cost of the total SE effort, and 2) the relative benefit of the SE technical processes along with the ideal budgets for each. As the objective here is to determine the most advantageous SE methodologies (by finding appropriate methods, processes, and tools), these cost models serve as an underpinning to determining the bottom-line of SE (as a metric estimated beforehand, and experienced after-the-fact). The review of baseline models begins with the intent specifically to answer PRQ 1.

2.1 Baseline A: The Constructive Systems Engineering Cost Model (COSYSMO)

COSYSMO is a cost-estimating relationship (CER) for SE effort[25]. Effort in this case means an estimate of the person-months necessary to tackle the SE tasks for a given product or system. While Brooks [26] might question how directly person-

months maps to cost as a factor of currency per person-month, this measure of effort is the current best model for the cost of SE. As a CER, COSYSMO is effectively a regression on data. This data was collected from surveys of SE projects, and the regression has been updated multiple times[27][28][29][30]. Currently, the model consists of size parameters, effort parameters, a scalabilty exponent, a re-usability exponent, and a calibration factor to fit the data. The structure of the COSYSMO CER is based on a predecessor CER for software engineering, COCOMO[31]. The first version of COSYSMO from [25] was later critiqued by Valerdi according to several industrial lessons-learned [27], and has since been revised with a particular view towards re-usability in the SE and product development processes[10]. These revisions culminated in COSYSMO 2.0, which includes additional logic for re-usability concerns as reported by [10, Fortune et al.]. The COSYSMO models are a cornerstone of estimating SE cost, and they have been extended for Systems-of-Systems [28] and implemented in SysML for demonstration purposes [32]. The overall form of the model is:

$$PM = a\left(\sum_{size} x\right)^{e} \prod_{effort} y \tag{2.1}$$

Where the overall cost reported by Valerdi is PM, or person-months. In equation 2.1, x are the size variables, y are the cost or effort variables, e is the effect of scale, and a is an empirical fit variable. According to [25, Valerdi], the size term is the weighted sum $x = \sum_i^3 w_i \phi_i$ where ϕ_i is the value for each size variable at 3 levels: easy, nominal, and difficult (see [10] for brief summary of the equations)

COSYSMO 2.0 takes the form of:

$$PM = a\left(\sum_{s}\sum_{r} \eta_r x_s\right)^{e} \prod_{e} y \tag{2.2}$$

Where in equation 2.2 s stands for size variables, r for reuse type, η_r for the reuse

weight, and x_s for the weighted sum from equation 2.1, and e for the effort or cost variables, all of which are the same as before. The result then from Fortune et al[10] is a revised estimate of the person-months for a SE project according to the level of reuse for each of the size variables, scaled according to effort.

The equations 2.1 and 2.2 provide a basis for estimating the total cost of SE in product development. That is, a SE implementation occupied with producing a SE management plan could forecast the cost of the plan according to these equations. However, as the COSYSMO models are mixed empirical and subjective models, they rely on new systems with characteristics similar to those which have come before. If a system or development program has characteristics foreseen to be outside the bounds or context of the historical data, or if a new development process than those used in the historical data is applied, then the regression coefficients found as part of the work of Valerdi [25] and Fortune [33] will no longer be appropriate. For example, Valerdi et al 2007[27] provide evidence that COSYSMO must be re-calibrated according to new industrial data. As the model has continued to evolve since 2007, there may be less bias towards aerospace and specifically defense than there was, but such a bias may still be inherent in the open models available. Further, Valerdi et al 2007[27] indicate that COSYSMO serves to estimate parametric cost for the SE life cycle, but at which point in the life cycle? According to the authors, this point may be at the very beginning of project planning. However, other authors such as Madachy and Jacques[34] appear to recommend use of COSYSMO in an MBSE context after the Architecture Definition[3] process and before the Design Definition process, a place when the system model produced under MBSE practice should contain requirements, interfaces, algorithms, and operational scenarios for application to the COSYSMO system size parameters. In this case, the modeler might interpret the person-months estimate as being applied to the upcoming phases of SE effort to realize the system design. Unfortunately for the system modeler, Madni and Purohit[5] propose that the

impact of MBSE practice is greatest during the phases up to Design Definition, and that the over-spending of MBSE methodologies compared to traditional SE primarily occurs during these phases, with benefits assumed to occur later. If COSYSMO is being evaluated against a baseline model, this means that the modeler is not aware during early processes that they are inefficient in their system modeling effort, before the baseline is established, and they may be unable to change course in their SE practice. The apparent inability to monitor the SE effort, detect when it begins falling short from the planned effort, and take corrective action to ameliorate any potential cost or schedule overruns is a major issue for planning SE tasks in detail. Such modeling is not possible with COSYSMO alone, as CERs model lagging indicators with historical data. Therefore, COSYSMO falls short of PRQ 1 and PRQ 2 as it does not provide sufficient insight into the impact of the SE method. These issues are summarized in Gap 1.

Gap 1 COSYSMO models only provide insight into the overall cost of SE for planning purposes, not particular methods.

As the focus here will be on methods for validation processes, in fact, it may be necessary to inspect some form of effort measure both early in the life cycle process for requirements validation per Grady[35], and later in the life cycle for system validation per INCOSE[3]. In each case, the normal "validation methods" of test, demonstration, analysis, inspection, etc., are applied within a larger set of tasks, i.e. the method, which describes how the process inputs are transformed into process outputs. A good solution to Gap 1 will provide insight at the level of these aforementioned tasks, while also having a correlation to effort measures as calculated by COSYSMO. This model of a SE measure should therefore provide insight into the SE methodology during its utilization in support of PRQ 3.

2.2 Baseline B: The Systems Engineering Return-on-Investment Projects

While understanding an estimate of the cost for a SE plan is useful in the planning phase of a project, industry also needs a means to estimate what the bounds or targets should be for this cost. That is to ask, how much SE is the right amount of SE? The works which aim to answer this question are the Systems Engineering Return-on-Investment (SE-ROI) project. The SE-ROI project culminated with the dissertation of Eric Honour[2]. This dissertation lays out the combined details published in a series of conference papers and web postings over the preceding decade. It consists of a collection of common SE technical processes labeled a SE ontology, multiple surveys based on the SE ontology to collect data as to SE effectiveness, and analysis of the results of the survey data with curve fits and statistical tests. The overall result is a set of normalized estimates pairing "effort" against return on investment. The key result is that the historical optimum found according to the survey data for the percentage of SE effort as part of total project effort for maximal ROI is about 14%. Additional data is provided for each of the technical processes, and for metrics other than return on investment as well, such as notional "technical quality". Findings showed that SE improved some metrics up to a point like ROI, and others found no correlation, such as technical quality. The limitations of the dissertation and associated studies by Eric Honour[36, 9] include the absence of measurement for particular SE methods or methodologies, as these are all confounded in the data collected. Additionally, all projects for which data was collected were performed under the paradigm of a traditional, document-based SE paradigm, rather than MBSE. New works building on SE-ROI are seeking to determine what the trends are for MBSE[23], but these studies will not have any statistical power to detect differences arising from particular methodologies due to a small sample of projects — where SE methodology here is used according to Definition 1, not as presented by Gooden[23]. That said, the result

of these studies is improvements in cost estimation and planning for SE according to historical rules-of-thumb, as to what percentage the estimated SE effort should be of the total effort for maximal ROI. Of course, these works suffer from some of the same epistemological limitations as COSYSMO. By building empirical models for SE-ROI, new projects which have qualities outside the bounds of the historical data may not be appropriately estimated by the fitted coefficients of the models provided. Going beyond COSYSMO however, the SE-ROI literature does give better insight as to the extent that SE processes should be performed on traditional programs. SE-ROI depended on survey results - this raises the question as to how might SE be measured across technical processes, and can it be done more objectively according to the products and processes developed or used during the SE effort.

While he states that COSYSMO is biased towards a sub-optimal level of SE effort — where the historical data on which COSYSMO is fitted consists of projects which may have not used enough SE effort — Honour states in his thesis[2] that both his and Valerdi's[25] models should not be applied to new SE methodologies and environments. However, as these older cost models are all that exist, they are being applied to MBSE as in Madachy and Jacques[34] as well as Papke et al[37] according to SysML model measures, among others for the MBSE paradigm. This gap means that PRQ 2 and PRQ 3 remain effectively unanswered, as Baseline A (Section 2.1) and Baseline B (Section 2.2) are geared towards holistic planning efforts in the context of known or traditional SE efforts. These takeaways are summarized in Gap 2.

Gap 2 Existing fitted models for SE ROI and SE Effort are not applicable to new methods, environments, and paradigms. A forecasting capability which predicts the change in SE measures according to SE tasks is necessary.

For Gap 2 it is important to understand the qualities desired in a forecasting capability. In order to best support the object of PRQ 3 given the emphasis on probabilistic knowledge by Hazelrigg [24], this forecast may be formulated in the form

of a conditional probability. That is, given a measure Q at time t, the probability of the measure reaching future value Q_r is $p(Q)$ at a future time t_f may be first guessed in the standard form of $P(Q_r|Q) = \frac{P(Q \cap Q_r)}{P(Q)}$[38]. This standard equation means that such a forecasting model may need information on the probability for the current state of the SE measure, and perhaps to account for the affect of time on the conditional probability. Both Gap 1 and Gap 2 have thoroughly established that fitting CERs to historical data, while necessary for understanding past performance, is not sufficient for understanding future states of current SE tasks in arbitrary context.

2.3 Measuring the Systems Engineering Implementation

The cost of SE is just one possible measurement going towards answering PRQ 1. It is important from the business perspective in terms of the potential burden that SE places on product development. However, another important metric is time. Time contributes to the rapidity of the product development process; in private industry, faster time for development improves the odds of being first-to-market, while in government industries time is a factor for cost overruns, contractual penalties, and potentially national security. As with the final cost of SE, the final timing of the SE effort is only known after some activities have concluded. Instead, some means is needed for forecasting the likely progress of the SE effort. As seen above, there are issues in historical models of the effort measure resulting in gaps Gap 1 and Gap 2.

The INCOSE Measurement Primer [39] proposes the concept of controlling SE processes by first measuring resources, processes, and products in order to operate a feedback control loop. In this way, the purpose of measurement in SE is not just to inform decisions about system design but also to inform decisions about the project/program, and the structure of the SE processes utilized by the organization. A key purpose of this information is to "identify problems and to take action to limit the impact of problems"[39]. Furthermore, a set of leading indicators has been defined

by INCOSE for the purpose of managing risk in SE by enabling "course corrections" earlier in SE programs[39]. The primer encourages "every player in an organization to be involved in not only the collection and analysis of measures, but also in the use of measures to aid identification and monitoring improvement opportunities in every element of the performance of business processes"[39]. However, it should be noted that the emphasis here is on creating good measures which address actual concerns of decision-makers, rather than measuring anything, or just the easiest aspects of the project. The primer indicates that applicable measures should change with the system life cycle, and that there is no perfect or absolute measure. That is to say, that the primer recommends that practitioners formulate new measures relevant to their context to satisfy PRQ 1. Doing so completely, however, may leave new SE efforts without historical data of past performance.

In order to better understand the likely trajectory of a SE enterprise, Rhodes et al[40] establish a definition of leading indicators:

Definition 2 "A leading indicator may be a measure, or a collection of measures, that is predictive of future system performance before that performance is realized"[40]

The intent of these indicators is to provide an estimate of the likely future outcomes of a program, rather than merely reporting a status-to-date, so that actions may be taken to correct the course of the program. These leading indicators were developed as a collaboration between INCOSE and LAI over about a ten year period. According to [40] the leading indicators include:

1. Requirements Trends
2. System Definition Change Backlog Trends
3. Interface Trends

4. Requirements Validation Trends
5. Requirements Verification Trends
6. Work Product Approval Trends
7. Review Action Closure Trends
8. Risk Exposure Trends
9. Risk Handling Trends
10. Technology Maturity Trends
11. Technical Measurement Trends
12. Systems Engineering Staffing & Skills Trends
13. Process Compliance Trends

In terms of the utility and possible applications, the official guide goes into greater detail[41]. While [40] lists Requirements Validation, Requirements Verification, and Technical Measurement trends as being the most useful according to survey data, the 2010 guide goes into greater detail. In [41], requirements trends including volatility, validation, and verification can be predictive of review readiness. Additionally, work product approval trends are considered predictive of review readiness, because of how gate review preparedness is frequently defined by SE practitioners. Review action closure trends may provide an indication that the project is ready to move on from the event which generated the action items, review or otherwise. Meanwhile, system definition change backlog trends may indicate the need for more reviews especially for complex or interdependent changes. Likewise, interface trends including interface definition stability may indicate need for additional reviews, especially when the stability of these definitions is low. Similarly, lower-than-planned rates for requirements

validation and requirements verification may indicate need for further review, but also if the rates are higher than planned, attention should be given to the quality of the processes used to ensure standards are being met. This material from [41] leads to Observation 1:

Observation 1 Leading indicators regarding requirements volatility and work product status are highly predictive of review readiness, and are therefore key to measuring cycle-time.

2.4 Baseline C: Modeling Requirements Volatility

A key capability to improve the usefulness of leading indicators is to know what values a leading indicator should achieve over time – otherwise known as a trajectory – so that cost and schedule objectives are met. This planned trajectory of the leading indicator is then a requirement against which a development team will be working – deviations from this plan may require corrective action, as discussed by Roedler et al[41]. However, the means to generate these planned trajectories and trends is not obvious. As the leading indicators are new and often highly tailored to organizations and projects, there may not be much historical data for calibration. Additionally, much of the data may not be published out of proprietary concerns. Fortunately, Walworth et al.[42] provide a simulation-based technique for estimating the planned trajectory of a requirements volatility metric.

2.4.1 Baseline C1: Walworth et al System Dynamics Model

Walworth et al develops a system dynamics model based on the archetypes of a learning system and the rework cycle, as described in the systems dynamics literature they report[42]. The authors go through a model formulation process to determine which factors should contribute to the flows in the model, and a fitting process to determine

to what extent they should contribute. While many of their results seem to match intuition, in particular for work quality, where the the work quality that is optimal with respect to schedule is slightly lower than 100%, other results seem to be imperfect or do not match expectations in all circumstances. For example, while adding more staff does appear to be approach a limit for a maximum rate at which the learning process/requirements activities can be completed, every higher level of staff is faster than the next lower. Intuition would indicate that for a given problem, organization, and competency level(s), there should be an optimal quantity of staff, below which work takes longer than desired and above which the same occurs, due to "dis-economies of scale" as in [25] or as described colloquially in [26] as an inverse relationship between team inter-communication vs productivity, where team inter-communication drags productivity due to a combinatorial explosion of communication channels with team size. However, despite this shortcoming, Walworth et al[42] presents one of the best setups currently known for creating planned trajectories of leading indicators, and should therefore be considered a standard model for requirements status/volatility, even if there isn't much insight into the requirements tasks which must be completed. The result here is a gap formalized by Gap 3:

Gap 3 How do specific SE methods affect the Walworth et al requirements volatility/status model?

2.4.2 Baseline C2: Grenn et al Requirements Entropy Model

Meanwhile, an alternative simulation technique for formulations of the requirements volatility leading indicator includes a physics-inspired model, known as the Requirements Entropy Framework[43]. This physics-inspired representation is inspired by thermodynamics and information modeling, insofar as requirements engineering aims to maximize the quality of the requirements set. By formulating an measure of "entropy" applied to requirements, Grenn et al[43] also formulate measure of enthalpy

and work – that is, they provide a means to estimate the SE effort to obtain a given change to the requirements entropy state. By applying these statistical techniques, the authors attempt to replicate the bumps in status seen in historical requirements volatility data. This provides better estimates in several cases than their baseline comparison, a (linear) fractional method. Perhaps most importantly, they show how their computed engineering effort aligns with the data used in [25] for the COSYSMO model. While no regressions are calculated or presented, the scatter appears to be a close match. Visually, this is clearly a pro for the entropy technique; the text below assures the reader that any differences are not statistically significant, with a p value of less than 0.05 ($n = 35$). As in most empirical approaches to SE measurement, analysis, and modeling, the available sample size is low. While the physical analogy presented by this model appears trustworthy with a calibration result that matches COSYSMO, the input data to the entropy model is primarily characteristics of the requirements and their metadata (such as current quality, target quality, number of, etc). The level of insight achievable regarding what the team should do, what actions ought to be taken, is questionable, other than the estimate of person-months needed to achieve a desired quality state.

2.4.3 Baseline C3: Incorporating Requirements Volatility in COSYSMO

Also addressing requirements volatility, Peña and Valerdi[44] documented the effects to be modeled for integration of requirements volatility parameters to COSYSMO and extending Baseline A (Section 2.1). The study involved a mix of qualitative and quantitative assessment via literature and multiple surveys and focus-group style interactions with SE experts. Peña and Valerdi establish the a priori assumptions regarding causes of requirements volatility, phasing, and relative impact. These efforts culminate in a scaling factor added to the COSYSMO model according to how the requirements are modified. As with previous and ongoing iterations of COSYSMO,

the emphasis is on planning SE in bulk. Overall Observation 2 is derived as a result of the gaps and literature discussed in this chapter.

Observation 2 CERs fitted to SE practice are inherently SE methodology-independent. Deeper analysis is necessary to model the effects of particular SE methodology.

2.5 Summary of the Benchmarking Exercise

The benchmarking exercise identified 5 baseline models for expressing the cost of SE. Section 2.1 and Section 2.4.3 are variants of the COSYSMO family of models, a series of CERs based on a combination of data, expert knowledge, and Bayesian techniques for estimating calibration coefficients. As a CER, these models lack a clear view into methods, identified in Gap 1. Section 2.2 is the ROI family of models, which apply a more traditional regression approach on SE data to establish the optimal application of effort according to SE Life Cycle Processes. The authors of Section 2.2 generally claim that their models, as well as Section 2.1 and Section 2.4.3, should not be trusted for new methods and paradigms of SE, a claim captured by Gap 2. A review of SE measurement practice identified the utility of leading indicators for modeling the readiness for gate review and transitioning the project life cycle phase, both of which are important aspects of process improvement, and results in Observation 1. Section 2.4.1 establishes a model of the requirements volatility leading indicator, but its lack of depth on method is captured by Gap 3. Section 2.4.2 presents a variant model of the requirements volatility indicator which relies on a physics allegory and is a black-box representation of requirements engineering activities. The useful aspect of Section 2.4.2 is that it maps volatility directly to effort, as in Section 2.4.3 but is not a CER dependent on historical data. Overall, according to Observation 2 not enough view inside the SE Methodology is provided by these models; only PRQ 1 is answered as of yet, while PRQ 2 and PRQ 3 remain unanswered by the baseline models of SE cost.

3

MODELING SYSTEMS ENGINEERING METHODS

"On the other hand, as the traveller [sic] stays but a short space of time in each place, his descriptions must generally consist of mere sketches, instead of detailed observation. Hence arises, as I have found to my cost, a constant tendency to fill up the wide gaps of knowledge, by inaccurate and superficial hypotheses"

– Charles Darwin, *The Voyage of the Beagle*

The previous chapter established a series of gaps indicating that the baseline cost models of SE do not provide deep enough modeling of the SE methods which have been applied, summarized in Observation 2. However, Observation 1 establishes that indeed some models and measures are predictive of the goodness of a project or program, and could be used in guiding process improvement. A key measure here is requirements volatility. However, in order to understand in a quantitative sense how a proposed SE method affects requirements volatility, it is necessary to propose an analytical formulation of the method to tackle Gap 3. To that purpose, this chapter specifically aims to identify approaches from the literature which support such analytical models. This literature review includes the fields of Business Process Re-Engineering. Subsequently, a discussion of the process representation for translation to simulation is necessary for understanding the constraints on implementation of such models. To make the discussion concrete, specific processes such as Verification and Validation which affect Requirements Volatility will be defined.

3.1 Establishing the Objective

In order to support decision-making on SE methodology, it is necessary to show the impact to SE performance in terms of the proposed methods. These impacts will be relative to the assumptions and calibration of variables, but without the ability to tie SE methods to modeling and simulation then the decision capability will be limited to qualitative comparisons. In the baseline approach, the proposed SE project is analyzed to determine the overall cost information. Most empirical models — such as COSYSMO, COCOMO, and SE-ROI — fit coefficients in mathematical models to data from multiple SE programs. The specific methods employed in each program may have varied and no variables control for this variability, other than *process capability*. Given that all systems engineers are competent with the processes and methods employed in a program, the models from these studies would give no insight into the particular impacts of tailoring actions or redesign of SE methods. Slightly more detail is gained by employing inspiration from physics and social science in terms of baseline leading indicator models. The REF provides information on expected effort needed to increase the quality or maturity of requirements, but as it relies on physical inspiration through entropy, the details of how requirements quality changes for individual requirements is not a matter of concern — just the net changes in "entropy". Likewise, the learning system analogy for requirements leading indicators does not provide detail on how learning happens, or rather how exactly learning power and effort translate into a learning rate. With the proliferation of methodology, it is important to have quantitative measures in combination with qualitative comparison for decision-making, considering the difficulties of attempting quantitative comparison between Structured Analysis vs. State-Analysis, especially if a third (e.g. Property Model Methodology[45]) or fourth (e.g. Rational Unified Process[1]) methodology is introduced for comparison without any link to the ultimate

bottom-line. As a first step, a literature review will be necessary to investigate how the baseline models might be enriched to provide greater information towards specific comparisons of proposed SE methods. The overall investigation supporting such a comparison is formalized as Research Question 1.

Research Question 1 How can any particular SE methodology be shown to be measurably better than another?

The answer to Research Question 1 will depend on the use of SE measures.

3.2 Business Process Re-Engineering

While it is well known that SE is performed within a business context, not much literature thus far has treated SE methodology as a *business process*. However, there is a vast quantity of literature on the subject of business processes.

Research Question 2 How can simulation of SE processes be tied to leading indicator models for planning and project management?

Jahangirian et al's review of 281 studies of simulation in business and manufacturing identified the primary domains of application, popularity, and pros/cons of several simulation techniques[46]. Included in the review were discrete-events simulation (DES), system dynamics (SD), agent-based simulation (ABS), among other techniques. Importantly, in the domain of "Enterprise Modeling and Simulation," the authors found that hybrid models were the most frequent - that is, hybrid usage of SD for the strategic or management impact level, and DES for the process or shop level of the simulation[46]. In this way, hybrid simulation aims to illustrate "the impact of production decisions, evaluated using DES models,... on enterprise level performance measures"[46]. This technique can be used for "hierarchical production planning," in order to determine the appropriate process decisions for low-level "shop" activity

in the larger business context[46]. However, as Jahangirian et al [46] discuss, DES has a relatively low impact on simulation stakeholders due to the time required for data collection or use of notional data instead. The lack of stakeholder engagement, measured in [46] according to whether the studies relied on real measurement, could indicate a need for swapping DES with some other category of model for multi-level business process simulation. Overall, the takeaway from [46] is Observation 3:

Observation 3 Multi-level business process modeling could be used to gain insight on how to structure lower-level process in a SE "shop" on the high-level leading indicators.

Observation 3 provides the missing elaboration towards Baseline C1 in Section 2.4.1 and Gap 3. That is, by construction a multi-level simulation, the learning model from Section 2.4.1 can have its undiscovered rework and discovered rework, and knowledge gained, modified according to detailed method proposals. For example, Lyneis[47] discussed the application of System Dynamics models to the aircraft domain. The works of Rabelo et al[48] and Venkateswaran et al[49] demonstrate hybrid techniques. Specifically, Venkateswaran et al explain that "the SD model captures the production and inventory dynamics of the enterprise, which are dictated by the decisions made by the shop scheduler...The DES model captures the detailed operational procedures of the shop"[49], while Rabelo et al[48] provide useful illustrations of what this combination means in practice. These hybrid techniques as discussed by [46] combine modeling and simulation methods, such as a DES, providing input to a SD model. Other combinations may also be possible. In all cases, effectively the translation to this domain vis-à-vis PRQ 2 is to create a representation of SE methods as if they were a "shop floor". Inherently, there is a need to ask whether this shop floor or any other metaphor is appropriate, and this is formalized as Research Question 3.

Research Question 3 How should the SE methodology be represented so that it

can be simulated?

The assumption for Research Question 3 is that simulations of SE methods which treat the tasks as the underlying model, such as happens for DES, might be able to be combined with Walworth et al's SD model.

3.3 Representing Processes and Methods

Besides simulation, it may be necessary to represent a process in such a way that other statements, such as proofs, can be substantiated. These statements are enabled by specific representation of processes and methods. If this representation of processes and methods can also be used to simulate the intended activity, then even better - it may be possible to use such a simulation in concert with models for leading and lagging indicators. When tailoring a SE methodology, a key outcome ought to be precisely how SE should be performed, and clear representations aid this effort. This section seeks to clarify means for representing processes and methods for SE, including how formal methods should be applied, as well as the options for semi-formal representation. According to Broy et al 2016[50], it is possible to represent the same meaning, or semantics, by different languages which each operate at very different levels of abstraction: these process representations even at the semi-formal level may blur the line between modeling and programming as discussed in Broy et al 2016.

3.3.1 Formal Mathematical Approaches to Representation

Formal techniques for process representation enable the derivation of theorems, application of proof assistants, and provide the basis for generalization and hypothesis generation for experimental verification of understanding. Generally, these formal methods are only lightly applied to the field of SE, primarily in the domains of cyber-physical and safety-critical software-intensive systems. However, through a broader

lens, SE rests on systems science[12] for its validity beyond governmental and organizational authority, in terms of the engineering disciplines. Several such theories have been put forward as to the mathematical formulation of SE. These theories are important as they have served to formalize the traditional practice of SE and begin the movement towards model-based SE, often understood to be the practice of SE via models representing SE products and processes. However, the active use of mathematical theories of SE in applied model-based SE does not seem to be very common. While there are some attempts at formalization in terms of set theory and category theory[51, 52], usually model-based systems engineers focus on the practical use of languages for model-based SE in their domain of application, and occasionally on formal representations of mathematics embedded within those languages for specific purposes (e.g. model transformations, transformation to particular simulation paradigms and formal algebra, etc). Two theories in particular seem to rest above the usual practice: 1) Wymore's Model-Based Systems Engineering and 2) Broy's Specification and Development of Interactive Systems. While these two works originate from different fields, they converge aspects of formal system representation. Here, a short comparison will be made along with some conditions necessary for application to practical simulation.

Introduction to Formal Process Comparison

This comparison will begin with an overview of the SE process and definitions given by Wymore. Broy does not provide a similar framework for the process of SE, though recent works such as the artifact relations [53] or the assumption/commitment contracts [54] describe activities which must implicitly occur within such a process, such as managing requirements traceability to architecture descriptions. This discussion will conclude with simulation-oriented off-shoots of the theoretical development.

The Wymore SE Process

Wymore provided a first complete synthesis of his theories from 1967[17] with application to the system life cycle in his book Model-Based Systems Engineering (MBSE)[16]. The purpose of the newer volume was to provide a systems-oriented language in the formalism of set theory and to demonstrate its application in the refinement of system architecture. Wymore's motivation primarily originated in a desire to show how "Systems engineering must start 'above' the level of physics"[16, p.1]. Even more importantly, Wymore[16] defines the discipline of SE according to what it ought to do:

- *Define the Need* - "to develop statements of system problems comprehensively,... without confusing ends and means,... without confounding the abstract and the concrete, without reference to any particular solutions or methods"[16]
- *Decomposition* - "to resolve top-level problems into simpler problems"[16]
- *Synthesis* - "to integrate the solutions to the simpler problems into systems to solve the top-level problem"[16]

Of principle importance, according to his defining that "Systems engineering is the intellectual, academic, and professional discipline the principal concern of which is the responsibility to ensure that all requirements for a bioware/hardware/software system are satisfied throughout the life-cycle of the system,"[16] Wymore focuses his theory on what the products of requirements analysis, that is the requirements which concern the system, ought to be.

Stating Requirements

For his approach, Wymore defines 6 categories of requirements:

1. "input/output requirements,

2. technology requirements,

3. performance requirements,

4. cost requirements,

5. trade-off requirements,

6. and system test requirements"[16, 19]

Using such a categorization of requirements may "avoid stating the problem in terms of a preconceived solution or class of solutions"[16]. This in particular is important as "When a solution is chosen in advance, the real problem is never stated, and *the solution fails because the world is not seen as it is but as the problem-solver wants it to be* so that his or her solution will be the correct one"[16] (emphasis added). Additionally, Wymore defines what it means to design: "The statement of a system design problem consists of explicit definitions of [the six requirements]. Collectively, these requirements are called the system design requirements" and a single requirement can be used for each category as "for any system design problem, the requirement in each category, no matter how complicated, is represented by a single system theoretic construct"[16]. With these preliminaries, it is then possible to begin establishing theorems and comparisons between systems theories.

Input and Output Requirement Comparison

The system theories first become comparable at the level of their proposed theorems, especially regarding the fundamental identity and definition of *systems*. For Wymore, the input/output requirements is defined because "every system of interest to SE accepts and processes inputs and produces outputs"[16]. Wymore defines that different kinds[19] of input signals are accepted on specific input ports and likewise output

signals on output ports. However, more importantly from the system theoretic perspective, Wymore defines "a trajectory is any function of time" and further "An input trajectory is a schedule or a time record of the way in which inputs can (do, will, might, ought to) arrive at the input ports of a system. An input trajectory can be thought of as a history of the input experience of the system"[16]. Additionally, "An output trajectory is a schedule or time record of the outputs produced by a system at its output ports over a period of time"[16]. These definitions are considered of high importance as "the statement of the problem of the design of any system must begin with the definition of the inputs that the existing system shall accept, process, or survive and the outputs that the system shall produce"[16]. Specifically the input/output requirement is defined by $IO : (l, i \in I, tr_i(t) \in I, o \in O, tr_o(t) \in O, f_e(tr_i(t)) \rightarrow tr_o(t))$ where IO represents the input/out requirement, defined by a tuple consisting of l "the length of the operational life", $i \in I$ "the set of inputs to be accepted by the system to be designed", $tr_i(t) \in I$ "the set of input trajectories or histories", $o \in O$ "the set of outputs", $tr_o(t) \in O$ "the set of output trajectories", and $f_e(tr_i(t)) \rightarrow tr_o(t)$ "the eligibility function that matches outputs and inputs, or input trajectories and eligible output trajectories; the eligibility function limits, or specifies, the behavior of the system to be designed"[16].

With the definition of Wymore's input/output requirement, the first comparisons to the systems theory of Broy, namely, FOCUS, can be made. For Broy, systems and their components communicate via channels which carry messages/signals of defined types (e.g. sets) in a manner that is "directed, reliable, and order preserving"[55]. An example of a Broy specification is given in Table 3.1, which is the Broy Adder. The Broy Adder represents an adder system which adds together the values of the input stream as they are received (un-timed). Likewise, there is also a definition from Wymore, the system specification given by Table 3.2 which is the Wymore Adder (1). The Wymore Adder (1) has a set of states S whose values x are given by the

Table 3.1: Example un-timed Adder Specification from [55]

Input	$i_1, i_2 : \mathbb{N}$
Output	$o : \mathbb{N}$
$\sharp o = \min\{\sharp i_1, \sharp i_2\} \wedge \forall j \in \mathrm{dom}.o : o.j = i_1.j + i_2.j$	

Table 3.2: Example Adder Specification from [17]

$\forall t \in \mathbb{R}, x \in S \exists \zeta \left(\sigma \left(f, t\right)\left(x\right)\right) : \zeta(x) = \zeta(f(t^-)) =$
$\sum x_i : i \in I[1, n], x = (x_1, \ldots, x_n) =$
$\sum f_i(t^-) : i \in I[1, n], f(t^-) = (f_1(t^-), \ldots, f_n(t^-))$

admissible values available on input ports and admissible input states, and is further defined in real-valued increasing time. At each time increment, the Wymore Adder (1) delivers the output set as ζ which adds all $x \in S$. The authors in [55] also provide a means for modifying specifications to be timed, synchronous, and other restrictions, so the Broy Adder in Table 3.1 may be revised accordingly. A key difference in the notation which arises from [55] is that the notion of types is used in predicate formulae; however the base type is the set. While [17] and later [16] have no notion of type, they are thoroughly constructed on set theory, and therefore similar results should be expected for timed or synchronous adder specifications.

Clearly the differences between 3.1 and 3.2 amount to notation and the domains of application which are given in the example content of the books. A more detailed comparison in terms of architecture products could be generated between Chapman et al[19], as the prototypical practical application of Wymore, against Broy 2018[53]. While this particularly detailed comparison is beyond the scope needed here, it may be of interest for establishing formal relationships between SE products before first using ontology to formalize relations in otherwise semi-formal approaches — later, it will be seen that these are usually the SysML, UML, AADL, and other languages applied in practical MBSE. Here however, the application of formal methods must tend towards simulation for the comparison of SE methodology.

Zeigler's Representation of Systems and Simulators

In the latest edition of the seminal *Theory of Modeling and Simulation*, Zeigler et al[56] lay out a means for extending Discrete Event Simulation (DEVS) for general systems in notation largely shared with Wymore's early work[17]. Zeigler et al are first concerned with knowing whether a system is "well-defined," and then secondly if any combinations of those systems are "well-defined" - that is to solve for "closure under coupling of a class of systems"[56]. These problems are handled as part of the definition of an "Iterative System Specification"[57]. The iterative system formalism explicitly builds on the discrete foundations from Wymore[17]. In that sense, there is an Input-Output trajectory or trajectories which are addressed by the model. The first step is to apply an algorithm to the trajectories called maximal length segmentation, which aims to identify "atomic components"[56]. The "atomic components" represent individual input-output functions in the same sense as Wymore's systems [17, 16]. The objective then is to reconstitute these objects in a "composition of component systems" which results in the desired output trajectory simulation given input trajectory stimuli[57, 56]. Via "proof of closure under coupling," Zeigler et al demonstrate how to construct valid systems composed of atomic systems of all subclasses - DEVS, but also discrete-time systems specification (DTSS) and differential equation system specification (DESS), and coupled DEVS-DESS[56]. Note that coupled DEVS-DESS may provide the theoretical foundations underlying bi-level hybrid business process simulation seen earlier in the development of Research Question 3. In Muzy et al, closure under coupling is used to model neuron behavior[57] according to activation models for DEVS[56]. Specifically, Muzy et al[57] detail a means of applying an "iterative system specification" based on a formal definition of a general input-output system to construct arbitrary discrete events simulations, specifically applied to a spiky-neuron simulation with mixed continuous time and discrete-event characteristics. This approach blurs some of the line between the formal mathemat-

ics and the programming language, as particular algorithms are necessary to realize the approach in application. Its ability to do so comes from the application of theorems such as closure under composition. However, the example is clear: the general input-output system is translated to simulation through the iterative approach. For well-defined processes which are compatible with the maximal length segmentation algorithm, this technique could be used to construct hybridized discrete events, discrete time, and/or continuous time models according to [57] and [56]. According to [56, Zeigler et al Ch. 10], the iterative system specification manifests by "[combining] Effective Computation Theory (as embodied in the Turing machine, for example) with Wymore Systems Theory (1967)". With the perspective of a general system as a process translating inputs to outputs, this means that such iterative system specifications may be applied to the processes and methods of SE should their constituent tasks be represented as such. Considering that recent simulations of Verification, Validation, and Test procedures as in Sudol and Mavris[58] (discussed later in this chapter) have been based on the discrete-events paradigm, the applicability of the general system and iterative system specification for discrete events simulation should not be surprising. However, application of the iterative system specification construct requires custom modeling and the creation of custom simulation environments for these models; there may be other solutions for certain aspects of these techniques, such as capabilities to demonstrate closure under composition.

B-Specification Language: Applications to Simulation and Compatibility

While a much larger section could be devoted to the B-Specification language, the purpose here is primarily to discuss the application of this language to deducing translator or adapter components and proving component interoperability, in a similar sense to closure under composition as in Chouali et al[59] or even embedded within what will be termed semi-formal techniques in the subsequent section by Mouahker et

al[60]. Focusing first on Chouali et al[59], it should be noted that this technique applying "design-by-contract" has been recently extended in mathematical terms by Broy 2018a[54], which provides much greater mathematical detail to rival the B method in terms of Broy's FOCUS system specification technique. However, the work by Chouali et al remains interesting because of its application. Specifically, Chouali et al provide an interface data model with pre- and post-condition constraints on interface operations via the Unified Modeling Language (UML) and its attendant Object Constraint Language (OCL), which may be translated by transformation rules to the B method theorem prover Atelier B for formal verification[59]. In as much as the authors are using an intermediary in front of the theorem prover, this could be considered a preview of the next section; yet the use of an intermediary such as UML does not entail the use of theorem provers, and thus this technique is organized here as a formal method. Specifically, the theorems which the authors aim to prove are constructed in terms of interface data model compatibility, with compatible required and provided operations, and for each operation that applicable pre- and post-constraints are valid (true) under composition of components via their interfaces. The authors demonstrate in a software system example for hotel reservations that they can prove a potentially off-the-shelf component meets the requirements levied by other existing interfaces in the system. The mathematics of the pre- and post-conditions, implemented in OCL with transformation to B-specification, should be compatible from Chouali et al[59] to the more recent techniques of Broy 2018a[54] with assumptions and commitments. While Broy 2018a[54] provides a pure mathematical example of a car door architecture with assumption/commitment contracts, application to translate the mathematics for machine use would still be needed. As the pre- and post-conditions techniques are highly similar mathematically if not identical via reliance on the same concept of refinement, the technology presented by Chouali et al[59] remains interested from the perspective of applying these sorts of proofs to actual architecture description. Furthermore, the

B method technique has many extensions, including the design of adapters between components by Mouahker et al[60]. These adapters are relevant as in the design of SE tasks or methods, it may come about that only a partial definition is known to fulfill a SE process; creating valid adapters could enable the insertion of placeholder tasks in simulation models for the design of SE methodology.

3.3.2 Semi-Formal Approaches to Representation

In Contribution [61, Paper A] and Broy et al 2016[50], the distinction between a "semi-formal" and a "formal" language largely revolves around the extent to which the product described in the language can be translated into a computer program. Other properties of these formal languages may be of interest, such as their applicability to theorem proving; however, the main interest regarding transformation techniques for modeling and simulation in MBSE languages is to take a less formalized model, and through some transformation rules produce a new model which is capable of computation. This idea loosely corresponds to the description of formal methods given by Broy et al 2016[50], but the main clarification would be that the formal language is primarily concerned with a precise definition and use of a specification. The distinction they provide between these formal languages and less-formal programming languages boils down to a specification giving the *what* of a model, while the programming language gives the *how*. Less formal representations, noted by Broy et al 2016 as including the Unified Modeling Language (UML) and the Systems Modeling Language (SysML), are very popular due to their visual representations and lack of need to work directly with formal types[50]. Despite the risk of "a discussion about precise semantics (what do two boxes with a line in between mean?) [turning] a project meeting into chaos,"[50] this risk may be worth taking if it means agreement on a language or set of languages with similar grammars for application of MBSE. This is a fundamental assumption of Contribution [61, Paper A] and Contribution

[62, Paper B] which describe these languages in greater detail.

Adding Formality to Semi-Formal Languages

With concern for the simulation of processes or tasks, it is necessary to consider how a representation language like SysML can be used to communicate about the process through visual architecture products, and also generate simulation models which provide numerical results. While Contribution [61, Paper A] provides an overview of some of the routes through which the latter is possible, there are some capabilities which are more relevant towards process-based simulation. These capabilities will be summarized here, by covering an overview of recent transformation discussions followed-up by an example in the DES domain.

A Brief Discussion of Model-Transformation Capability

The conference paper by Cole and Jenkins[63] is of particular interest beyond the potential of translated SysML models into requirements text due to its discussion of transformation, ontology, and patterns. An example of this discussion is illustrated in Figure 3.1 with additional markings regarding this thesis author's opinions on other transformations in the literature. There are a few key aspects of Cole and Jenkins to take away: the limitation of the use of SysML to a particular pattern or template, the implementation of transformation rules through software infrastructure and fast searching algorithms, and a target meta-model which provides an execution model for computation, in this case Mathematica[63]. It is also possible to break the process of transforming the SysML source model to the target into a set of steps for easier construction of the relevant software as shown by Cole and Dinkel[64]. What would be most appropriate in this case is to specify some portion of the SE method in the MBSE language, and then via a query and/or set of transformation rules produce a simulation model formatted for use with an appropriate simulator.

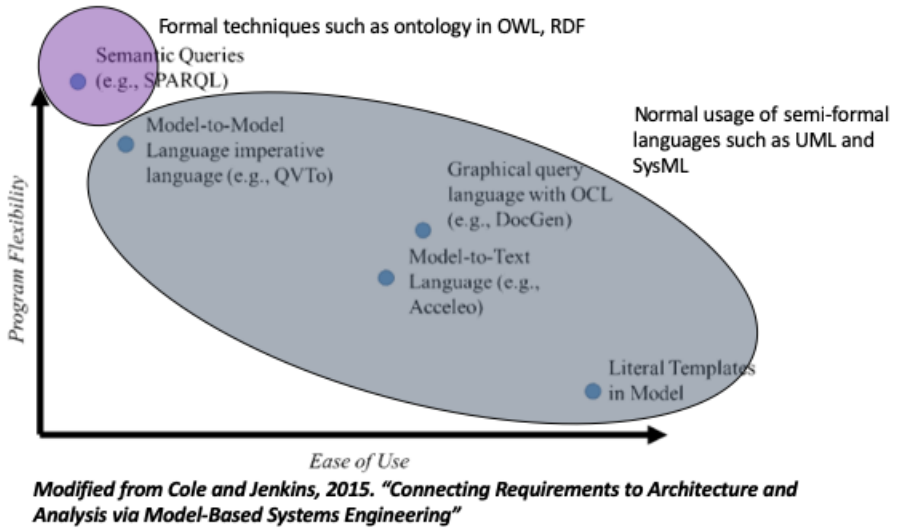


Figure 3.1: Model Transformation Difficult Mapped to language types based on[63]. Regions are colored based on typical domain of application.

Transforming SysML to DES

One particular work has already demonstrated a capability to transform SysML to DES using the some of the principles described above, albeit with different implementation and infrastructure. The techniques above define what might be considered a domain-specific language (DSL)[50, 65] overlaid on the SysML syntax. Sprock and McGinnis[65] define a DSL of this kind to translate SysML to DES for discrete-event logistic systems. For Sprock and McGinnis, the SysML model describing the *ProcessNetwork* is the source model which is translated to DES. This particular paper is light on how exactly the "transformation engine" is constructed, which is to be expected due to the authors' discussion about the lack of "a complete methodology and supporting tool-chain" for discrete-events logistic systems[65]. However, this discussion on process representation can be concluded by posing Research Question 4, extending Research Question 3:

Research Question 4 How should a transformation be constructed for simulation

of SE methods via DES?

3.4 Methods Describing Validation

The validation process is usually further defined as "X Validation" where X is any product created during the SE process over the system life cycle: requirements, drawings, design data, system elements, and even the system itself[3]. In each case, the objective is to confirm whether the product meets stakeholder expectations. There is some controversy in the SE community regarding the precise definitions of Validation and Verification[15, Grady discusses at length starting at p. 14], especially in the context of any particular "X." For example, the Aerospace Recommended Practice 4754A for Aircraft Development Assurance suggests that requirements validation should show that the requirements are complete and correct according to stakeholder interests through analysis, testing, etc[66]. There is significant overlap between this concept and that of requirements verification from INCOSE, stating that it is "to check the application of syntactic and grammatical rules and characteristics... such as necessity, implementation-free, unambiguous, consistent, complete, singular, feasible, traceable, and verifiable"[3]. That is to say that according to the SE Handbook, requirements verification establishes that requirements are verifiable, but does not verify conformance to them. Additionally, the handbook and ARP 4754A are referring to similar activities using different terms. However, SE educators take a slightly different approach which adds detail to the validation process. Some techniques for validation on this basis are described in Contribution [62, Paper B]. Specifically, Grady's technique would involve the improvement of understanding of requirements[15], and that this process will involve adding, modifying, or deleting requirements throughout, including checks for syntactical correctness[35], in place of the INCOSE requirements verification. Grady's validation process is shown in Figure 3.2. Martin also provides input, describing a validation process which "ensures that the requirements are

consistent and complete with respect to higher level requirements"[4]. In all cases, validation methods are similar to verification methods, but the basis of comparison for the results of X are different. Usually, the validation process is concerned with stakeholders, but it is also concerned with feasibility and the design team's understanding of the problem — in all cases, summarized in some form of the question "are we building the right X." The overlap identified in Contribution [62, Paper B] with trade study and technology investigation techniques through Robust Design Simulation (RDS) and validation also applies to the learning system analogy of Walworth et al[42]. As validation proceeds, requirements will be quantified with feasible values and the engineering team will gain better understanding of the solution space.

In a concrete sense, Sudol and Mavris demonstrated a DSM-based DES technique for modeling VV&T re-work costs as applied to the space shuttle main engine development[58]. This work demonstrates in principle the idea behind Research Question 3 and specifically gives one means of simulating an historical SE method. However, the proposal here seeks to compare different methods for SE processes; with the same process, that is a different DSM representing the method-as-a-process, i.e. the method-process inversion discussed by Martin[4].

3.5 Verification and Validation Procedures for Computer Simulations

There are a few levels of model and simulation which are concerning in terms of validation for this work:

- Engineering disciplines and analytical models for computing system characteristics
- System models authored in MBSE languages
- Process models for simulating the SE process/method tasks

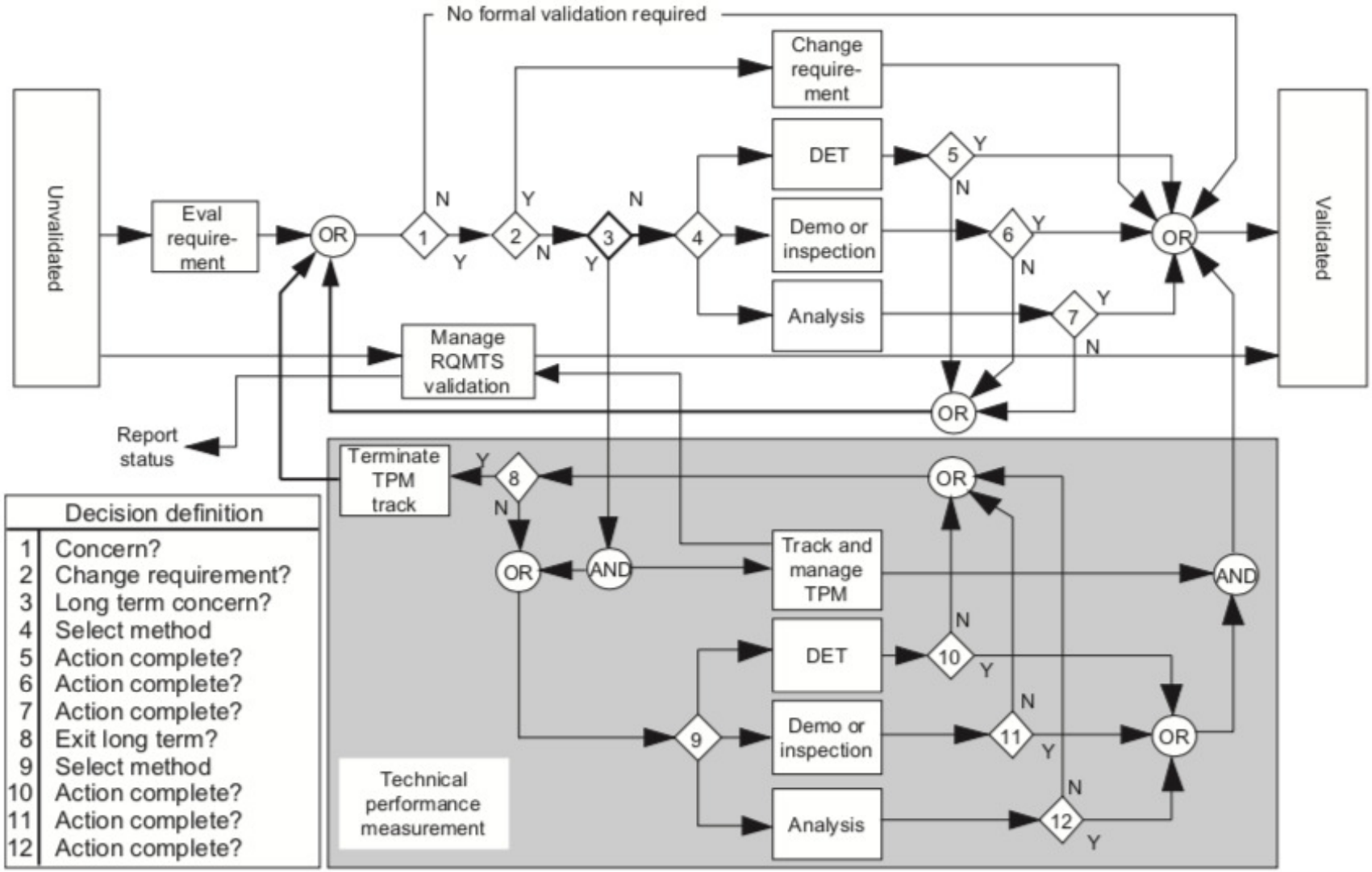


Figure 8.15 Item requirements validation process.

Figure 3.2: Grady's validation process from Grady[35]

Each case above is a computer model which may be subject to some sort of simulation and which may require a verification and validation procedure as evidence for its completeness and correctness. This is especially important if the model is to be used as part of a decision-making exercise or is part of a safety-critical system. Fortunately, Sargent 2010[67] elaborates on how to accomplish these tasks.

3.5.1 Review of Model V&V

According to Sargent[67]:

> "Model verification is often defined as "ensuring that the computer program of the computerized model and its implementation are correct" and is the definition adopted here. Model validation is usually defined to mean 'substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model' (Schlesinger et al. 1979) and is the definition used here"[67]

Verifying the implemented model matches the algorithms intended; this is fulfilled by unit tests of functionality (e.g. was Newton's method implemented as Newton's method?). On the other hand, the validation of a model is to exercise it and provide a judgment as to whether it is accurate enough for its purpose. Sargent further elaborates that

> "A model should be developed for a specific purpose (or application) and its validity determined with respect to that purpose. If the purpose of a model is to answer a variety of questions, the validity of the model needs to be determined with respect to each question"[67]

A system model in the form described by Wymore or Broy is a model which can be subjected to simulation; these models are thus subject to model validation in the

sense described by Sargent. However, since the purpose of these models is to match all input trajectories and output trajectories for a system, the complexity of the testing requirement could become quite high. A real concern arises as to whether the system modeling activities may be un-validated. Architecture evaluation methods[68, 69] do not necessarily map to the model purpose. For example, measuring the number of edges between architecture products and requirements may indicate the supposed coverage of the requirements by the model[69], but this figure is only relevant if the model is capable of producing some evidence about the status of the requirements in terms of the architecture (i.e. are the requirements which are covered satisfied and to what extent are they satisfied?). The system model itself must be subject to validation to be correct. This means that asking how to validate the system model is a critical question to conducting MBSE.

According to Sargent,

> "Conceptual model validity is determining that (1) the theories and assumptions underlying the conceptual model are correct and (2) the model's representation of the problem entity and the model's structure, logic, and mathematical and causal relationships are 'reasonable' for the intended purpose of the model"[67]

As a system model is necessarily conceptual (i.e. as expressed in a set-theoretic modeling language, or in SysML, or in AADL, etc), system modelers ought to be concerned with this notion of validity of these models.

3.5.2 Validation of SE Models

As far as correctness of grammar and completeness (INCOSE validation checks especially for requirements)[3], the MBSE literature does provide some insight. According to Jackson et al[70], for MBSE approaches the validation of the system model (is the model capable of accomplishing its purpose) are key to understanding validation mea-

sures of the system requirements. Jackson et al[70] establish procedures for evaluating "model audits for correctness and completeness" implicating a validation activity per the INCOSE handbook[3] and further supported by one of the authors' SE questions: "Where did this requirement come from, who has looked at it, what reviews and quality checks has it passed?"[70]. Since the outputs the audit measurement are quantities like number of requirements TBD, TBR, and so forth, Jackson et al have established a means of providing based measures useful for leading indicators from a system model. However, in terms of validation, they are not measuring the performance of the system represented by the system model in some context; instead this is a validation of the work done by the SE team. With the tooling provided by Jackson et al, system modelers know their progress towards a correct and complete system model according to the grammar, rules, or criteria which have been asserted on the system model. What is still lacking is validation in the sense of Wymore: does the system model have correct input-output trajectories? Or, in the terminology of Sargent, does the system model exhibit operational validity?

3.5.3 Model Validation and Representations

In the practice of system modeling, a key question is the extent to which the conceptual model, in the terms of Sargent, not only provides a specification for the computerized model but is the computerized model. This exact issue is discussed by Broy et al 2016[50], where the authors demonstrate how modeling languages may be equivalent to programs and that there is not a fine line distinguishing the two. Sargent acknowledges this overlap regarding verification of the computerized model[67], in that "specialized simulation languages" may provide both the conceptual definition and the computerized simulation. For Broy et al 2016, such a specialized language is a Domain-Specific Language (DSL) whereby conceptual constructs are translated into programs directly via rules defining the DSL grammar.

For requirements validation, a key concern is whether the quantities specified are feasible. Contribution [62, Paper B] asserts that RDS might be able to provide this information. As such, computer models of performance are needed with valid model representations as part of the SE task. If this task is occurring under the MBSE paradigm, then there ought to be a centralized representation in a system model. If the system model is valid and the requirements are feasible, the assumption then is that the real system might have some sort of operational validity. If a real system exists and is operational, then the requirements were valid to begin with.

In effect, this process describes a loop of validation. Requirements are valid because they are feasible; feasibility is ultimately determined by what physical systems can be realized, and operational systems are necessarily realizable. This is a bootstrapping problem, as requirements feasibility can be established via RDS. RDS requires surrogate models. Surrogate models are valid proxies for computer models but maintain model representation only on the basis of inputs and outputs of the computer model. However, when the properties of these models and representation artifacts change, the thread of the logic between the models might break. If the concept or mathematical specification is modified, then the surrogate model no longer represents the computer model.

Here begins a jumping-off point to further investigate means of maintaining a chain of logic from the system model which may undergo validation processes, to other models which are used to perform validation. Research Question 5 formalizes the intent of this new line of questioning, to determine a means of tightly coupling the system model with the engineering models used for validation.

Research Question 5 How should a DSL for RDS activities interface to M&S, uncertainty, and other infrastructure from a system model?

According to Research Question 5, an answer is needed as to whether a DSL be created which can apply uncertainty in the cases where model representation is

known to be lacking or equivalently where the maturity of the system model elements are low and thus requirements volatility is high? Thus when the computer model is generated, uncertainty parameters are provided regarding the aspects of the system at low maturity. For example, preliminary design may seek to optimize the structural layout but without a robust solution, this optimum will not be correct due to low-maturity/lack of model representation for fixtures and other similar issues. For a product-based formulation this would necessitate the creation of surrogates of the product-based modules, which are then valid within the bounds of the uncertainty sample. Requirements quantities selected within these bounds have a probability of success according to the priors. If a detailed design can be found within the bounds, then the preliminary design was a robust solution and the project may move forward without costly rework.

The above implies 1) a mechanism to inspect or test whether the validation approaches are keeping to schedules, and 2) a DSL for generating RDS on a product-based formulation of the multidisciplinary design problem. The next chapter will be dedicated to exploring the means of standing up a multidisciplinary design problem from a system model in support of Research Question 5. Inherently the system model must express criteria, objectives, constraints, and the like. In the style of Jackson et al[70] this is normal, as constraints precede requirements precisely for the reason of delaying decisions on the requirements quantities until the system maturity is appropriate.

3.6 Concluding this Voyage

Much literature has been discussed to this point which provides tools and techniques for modeling business processes, representing SE processes mathematically, modeling SE leading indicators, and SE effort. However, there are some key concerns which arise in terms of the validation of models and requirements which relate to

the tasks which are modeled by SD, DES, etc. For example, *why* are requirements volatile? If requirements engineering is a learning system, what is learned and how is it learned? If more functional process models are used to describe the tasks underlying requirements engineering, is there any set of tasks against which the learning can be mapped? Finally, Contribution [62, Paper B] established that RDS has a place in early-phase requirements validation. Through RDS the engineer determines which values of requirements have the best likelihood to result in a realizable system. To find the likelihood, surrogate models of the metrics against which requirements are to be posed must be produced. The tool which best suits the calculation of these metrics according to the system and physics of the engineering problem is multidisciplinary Design Optimization (MDO), which will be investigated in greater detail in the next chapter.

4

SUPPORTING ROBUST DESIGN SIMULATION THROUGH MULTIDISCIPLINARY DESIGN OPTIMIZATION

Must not whatever can run its course of all things, have already run along that lane? Must not whatever can happen of all things have already happened, resulted, and gone by?

– Friedrich Nietzsche, *Thus Spake Zarathustra: A Book for All and None*

4.1 Purpose

The purpose of this chapter is to expand on the idea presented in the previous chapter regarding a particular validation method, and more specifically how it related to ensuring valid systems and requirements. The previous chapter concluded with observations on the applicability of RDS in validation per Contribution [62, Paper B], and also the need for model validation which may be enabled by better coordination with a system model representation. With these observations, the present chapter seeks to establish further observations according to Research Question 5 which apply the previous work and research of this author, (extending the work presented in Contribution [61, Paper A]), while revisiting some topics briefly mentioned before such as model-transformation and DSL. To restate the research question which concluded the previous chapter,

Research Question 5 How should a DSL for RDS activities interface to M&S, uncertainty, and other infrastructure from a system model?

Some portions of this question are well-known to be answered in the literature, and come to a matter of implementation. An overview of this literature will be given.

Some basic principles will be presented, from which gaps and questions are established to enhance the state of the art.

4.1.1 Brief Description of RDS

A brief summary of RDS is given in Contribution [62, Paper B] based on Mavris et al 1999[71]. The idea behind RDS is that "the better design-point performance is traded for the superior off-design performance" of a "robust" solution[71]. A case in which this premise is understood is common in the design of aircraft. A commercial air transport might be optimized for the lift to drag ratio at a particular flight condition in cruise. However, the aircraft must operate in many flight conditions across its mission(s) other than the single point. Optimization to a point may limit the overall performance of the system in real conditions. RDS leverages probabilistic techniques such as Monte Carlo simulation on response surface equations of the design space to provide information on a "probability" of achieving a constraint target. At the core of RDS lie these response surfaces, which are constructed by fitting an approximation model to data. The data for the approximation model must be provided through some means: it could be experiment, or more likely it will be from engineering modeling and simulation known as "synthesis"; this synthesis activity is enabled by MDO to provide sized solutions at each point in the data.

4.1.2 Defining MDO

In the previous chapter, some specific MDO literature was given in the context of MBSE model management and manipulation, for the purpose of validating the system design and in the context of model validation. MDO, also known by other abbreviations such as Multidisciplinary Design Analysis and Optimization (MDAO), is "a methodology for design of complex engineering systems that are governed by mutually interacting physical phenomena and made up of distinct interacting subsystems",

as described by Sobieszczanski 1995[72]. In particular, Sobieszczanski describes how various categories of numerical methods, software/human interfaces, and optimization/search routines are applied to find values for design variables which meet the conditions specified by objective functions and constraints. Some detailed background on MDO is provided in this section. First, there is an overview of the techniques used in formulating MDO problems. Then, there is a summary of configurable MDO technology. A key component of configurable MDO includes the manipulation of a directed graph occasionally abstracted as a Design Structure Matrix (DSM), and some notes on DSM manipulation will be provided. Finally, efforts to formulate product-based MDO problems are described, including techniques for graph replacement and model transformation.

MDAO According to Balesdent

Balesdent et al[73] provide a survey of several Multidisciplinary Design Optimization (MDO) methods for the purpose of selecting the most appropriate methods for the problem of launch vehicle design. The paper discusses some traditional means of MDO, such as Multi-Discipline Feasible (MDF) or Nested Analysis and Design (NAND), Single NAND-NAND (SNN), and All-in-One, with references. For MDF, the main advantage is simplicity due to limited numbers of design variables and no need for system decomposition. An additional advantage is that each iteration provides a feasible design, even if sub-optimal. However, MDF may increase computational cost for coupled systems/disciplines, and computing gradients or changes in variables will requiring running the full Multi-Disciplinary Analysis. Another traditional method is Individual Discipline Feasible (IDF), or Optimizer Based Decomposition (OBD), or Single Simultaneous Analysis and Design (SAND) NAND (SSN), does not require running a complete MDA for each iteration. However, the optimizer must manage all coupling variables and the result between iterations may not be a

feasible design, until convergence is reached. An even simpler method is All-At-Once, or AAO, also called Single-SAND-SAND (SSS). This method simultaneously solves the optimization problem and equations for subsystem analyses, so that the design is only feasible at convergence, and subsystem equations are handled as residuals or equality constraints. While it is simple, and can be performed for complex problems with simple equations, AAO is not applicable for complex subsystem problems. Besides these traditional methods, several multilevel methods attempt to improve MDO capabilities. This includes collaborative optimization (CO), Concurrent Subspace Optimization (CSSO), Multi-Objective Pareto Concurrent Subspace Optimization (MOPCSSO), Bi-Level Integrated System Synthesis (BLISS) and BLISS2000, Modified Collaborative Optimization (MCO), Analytical Target Cascading (ATC), Discipline Interaction Variable Elimination (DIVE), and Dynamic Leader Follower (DyLeaf). The authors discuss some properties and views in the literature regarding these optimization strategies. They conclude with a discussion of the applicability to the launch vehicle problem, and the possibility of a physical-decomposition orientation for organizing disciplinary analyses to improve the MDO process. The technique mentioned at the end of Balesdent's review, SWORD, will be discussed later.

Re-configurable Formulation of MDO

The material described above by Balesdent, the MDAO architectures, would be described by Pate, Gray, and German 2013 as solution procedures[74]. However, Pate et al generalize these solution procedures through series of graphs and algorithms to manage the configuration of MDO problems. In particular, they describe a maximal connectivity graph (MCG) of "all possible interconnections between analysis tools", followed by a fundamental problem graph (FPG) which represents analysis tools selected for some specified design problem. The authors introduce a series of algorithms for pruning the MCG based on its characteristics to result in the FPG depending on

the input variable nodes, analysis blocks, and expression blocks provided in the MCG. The authors argue that their contribution is more useful in computation than XDSM due to consistent data structures for the FPG and problem solution graph (PSG), at the cost of reduced visual appeal. Pate et al[74] then apply the MCG and FPG algorithms to an aircraft conceptual design problem to illustrate the pruning of the graphs when multiple fidelity levels of duplicate analyses are available. Pate et al provide a key capability that is widely used in the current state-of-the-art and supplemented by the process-integration and design optimization (PIDO) tools they discuss in their introduction, some of which are also addressed in Contribution [61, Paper A].

Refinement of the PSG

Building on Pate et al, several European authors have in recent years made refinements to the PSG formulation and gone beyond the original work. This section reviews the model-based engineering and MDAO capabilities under development at TU Delft, DLR, and other European locations under the Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts (AGILE) Project. Specifically, Gent et al 2017 [75] present a series of steps for problem formulation, workflow creation, and execution that go beyond Pate et al. For Gent et al 2017, problem formulation has three steps - formalize, debug, and manipulate. For the authors, formalization is to create a database of all analyses tools, with inputs and outputs relationships recorded in the database according to a schema like CPACS (described below). Their debug step is to translate the database information into a directed graph data structure in order to inspect the connectivity between analyses. Finally, a pruned directed graph based on Pate et al is manipulated to produce a desired MDAO architecture and workflow for executing the MDAO problem.

The idea of a maximum connectivity graph is replaced in the database of Gent et al 2017 [75] with a Repository Connectivity Graph. As in [74], steps of node removal,

node contraction, and node enrichment are performed on the Repository Connectivity Graph to produce a Fundamental Problem Graph. Removal eliminates unused nodes; contraction abstracts sets of analysis nodes with no feedback whether sequential or parallel; enrichment uses additional metadata in the database about variables to inform the design problem, such as which variables are design variables, bounded, constrained, etc.

The step of workflow materialization focuses on integrating the open-source graph data generated during formulation with the proprietary PIDO tool chosen by the authors. This tool is not unlike ModelCenter as discussed in Contribution [61, Paper A]. The workflow materialization for Gent et al 2017[75] is essentially the PSG for Pate et al[74]. The final step appears to be the execution of the materialized workflow, but it is not explicitly stated in Gent et al 2017.

The analytical system is able to generate IDF and MDF MDAO configurations of the Sellar problem. Gent et al[75] were also able to test an aero-structural wing design optimization problem. They report a particularly sizable aero-structural problem, with 28,196 nodes and 37,509 edges in the RCG of the database[75].

Meanwhile, Gent, Rocca, and Velhuis [76] provide even more detail on the MDAO graph execution system, specifically on the graph data component KADMOS or "Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System". The authors claim that this tool enables "quick formulation, reconfiguration, and execution of MDAO workflows using distributed and heterogeneous sets of analysis tools"[76]. A key issue that they aim to address is the size of MDAO problems, specifically related to determining feasible data flows. This issue arises due to combinatorial explosion, as "to automate the system of disciplinary analyses the connections between those analyses have to be defined down to the smallest detail before a functioning system can be implemented"[76] for parameter connectivity. They claim that the KADMOS tool will formalize, inspect, and manipulate the specifications of MDAO

architectures. To accomplish these goals, Gent, Rocca, and Velhuis used technologies such as a "central data schema, knowledge-based techniques, graph-based approach, [and simulation workflow] software packages"[76].

One technological motivator for Gent, Rocca, and Velhuis is the use of a central data schema, or more specifically a centralized data store with common data model, which promises in theory to decrease the total number of software interfaces from $n(n-1)$ to $2n$. The authors used an XML format based on the Common Parametric Aircraft Configuration Schema (CPACS)[76]. As a development requirement on disciplinary analyses, all disciplinary analyses must document and implement their interfaces according to CPACS for the larger project (AGILE) around KADMOS[76]. This is an organizational means of enforcing conformance with the central data model necessary for the graph-based approach. After the tools are developed, then the KADMOS tool can create an overview structure that begins the MDAO architecture creation process.

The second motivating capability for Gent, Rocca, and Velhuis is knowledge-based technologies[76]. The key aspect of this is an enriched database. Features implemented with this database include an XML schema definition likely used to check data conformance, meta-data information files, input files and output files, all formatted in XML with only elements conforming to the schema[76].

KADMOS has two digraph types: data and process[76]. The data graphs in KADMOS are digraphs with two sets of labeled nodes: functions and variables. Functions represent some software to be executed while variables represent data from the central XML database. KADMOS process digraphs consist of a single node set for functions, with edges being process flow direction. Digraph sub-types largely align with [74] however some additional configurations for describing MDAO problems, and for describing the executable simulation workflow in the target environment, are provided. Metadata is computed on data digraphs for pruning and manipulation as per the

algorithms in [74]. Role assignment for MDAO is performed over variables nodes, and then the functions must be manually sequenced - however there are unspecified plans for automated sequencing routines.

The optimization description capabilities from Gent, Rocca, and Velhuis [76] include testing tools, converging a multidisciplinary analysis, performing a Design of Experiments, running a MDF without convergence, running MDF with convergence, or running IDF. Algorithms for generating MDAO problem and data graphs differ based on which numerical convergence methods are chosen. As an example, the Sellar problem is addressed in KADMOS. Then, a wing design aero-structural analysis represented in the standardized XML database is converted to several description graphs for solving under the different available algorithms.

Gent, Rocca, and Velhuis [76] intend to add additional MDAO algorithms, additional MDAO tools, and completion of test case problems such as aero-structural wing design from Gent et al 2017[75]. The key supporting data-structure after describing a design in their description language is the graph representation of analyses and optimization for the characterization of the design. As a result of the representation of the analyses workflow, a DSM may be generated as a traditional presentation of the analytical framework[76].

4.1.3 Description and Role of the Design Structure Matrix

The DSM is a traditional SE visualization. It "provides a simple, compact, and visual representation of a complex system that supports innovative solutions to decomposition and integration problems"[77]. The DSM is often used to rearrange a system decomposition into subsystems, illustrate how subsystem relationships result in system behavior, and map external interfaces to system performance[77]. Browning's survey divides DSM into two general categories: static and time-based DSMs[77]. Browning organizes the categories as: static-component-based or architecture, static-

team-based or organization, time-based activity or schedule, and time-based parameter DSM[77].

Some combination of the DSM classes is expected for organizations, systems, and analytical processes. One additional note from Browning, with respect to the previous chapters of this thesis proposal, the recommendation for representing processes includes:

- "decompose the process into activities;"[77]
- "document the information flow among the activities (their integration);"[77]
- "analyze the sequencing of the activities into a (generally) maximally-feed-forward process flow."[77]

Finally, Browning identifies a role of DSM regarding MDO other than visualization, in that the clustering of the DSM carries implications for convergence of numerical methods[77].

Use of DSM in MDAO

In MDAO, usually a plain parameter DSM is insufficient to describe in a visual manner what the total flow of information will be for a given problem. This is because, as described above, MDAO requires information both about the parameter sets but also the workflow intended for solving the MDAO problem. An important finding in the literature is the XDSM [78], which "The [extended design structure matrix, XDSM] is based on extending the standard design structure matrix (DSM) to simultaneously show data dependency and process flow on a single diagram"[78]. The point of the XDSM is to display in one view the combined activities and parameter flow needed to setup an MDAO problem and solve it, otherwise known as an MDO architecture[78].

DSM in MBSE Languages — The Semantic DSM

Diagne et al developed a technique for DSM in MBSE languages like UML for system performance analysis[79]. The motivation for extracting an analytical DSM from the system model is to enable evaluation of the product family described by the system model[79], i.e. the universe of instances defined by the class structure. According to Diagne et al, "In this approach, the inputs of the DSM are component classes and link classes that allow adding semantic data"[79]. The matrices which result from parsing their class diagram model contains both the component linkages expected for a Component DSM, as well as notation for various -ilities regarding the performance of the system architecture. Each instance of the classes defined by the model in UML corresponds to a particular matrix. A key aspect of this work is the notion of applying set-based exploration of designs according to the UML model which was created. This aligns with works to be discussed below.

4.2 Role of MDO in SE

Being able to do MDO in support of RDS is a nice capability, but for the purpose of this thesis proposal the MDO capability must provide some benefit to SE practice. Therefore, some understanding of the role of MDO in SE must be established. In fact, the literature supports the argument that MDO and RDS serve to enable early-phase validation actions:

> "Thus, better systems engineering tools are needed to enable a more intelligent, thorough search of the trade-space during the conceptual design phase of a program. The methodology [for multi-objective MDO] developed in this paper provides just such a tool to help systems engineers make better up-front design choices to improve the system performance and reduce the life-cycle cost of future distributed satellite

systems."[80]

The systems engineering decision-making process is thoroughly described, including trade-space exploration, by Parnell[81]. However, the understanding of the role of MDO for SE from Jilla and Miller[80] is compatible with the intent of RDS. MDO is necessary to make better decisions about the system design in early phases of development. This also supports the conclusions about the direction of research from the previous chapter and from Contribution [62, Paper B] that MDO serves a role in validation actions in early phases to determine the feasibility of a design in terms of the current technology.

4.2.1 Argument for MDO Language

According to Ceh, "a [Domain Specific Language, DSL] should be developed whenever it is necessary to solve a problem that belongs to a problem family and when we expect that in the future more problems from the same problem family will appear"[82]. Ceh recommends that the DSL be specified according to an ontology. Regardless of the precise method of specification, it is clear that MDO activities will occur frequently, especially early in design. Therefore, as part of the SE model and model validation, a representation for MDO like a DSL may be desirable. This is especially the case if the SE model is expressed in an MBSE language which may be compatible with embedding or transformation to the DSL. However, this is not a new idea. In particular, Stephan Rudolph's lab from Stuttgart has produced several works which illustrate a variety of uses for a "Design Compiler" across several design problems based on design data in UML/SysML.

4.2.2 Modifying the Formulation of MDO Problems for Product-Oriented Decomposition

Balesdent 2016[83] elaborates on a product-based formulation for MDO of launch vehicles which also incorporates uncertainty. This represents a very different way of

view MDO problems than the usual disciplinary approach. While for each stage in the vehicle under design by Balesdent there is more traditional MDO sub-problem, the options for modifying the MDO procedure begin to open up with this new perspective. In fact, a product-based perspective may even enable deeper coupling between MBSE languages and MDO techniques. Additionally, more recent work has demonstrated the construction of object-oriented, automated, MDAO environments along a similar paradigm. Edwards et al 2018[84] presents such an environment, named the DYnamic Rocket EQuation Tool (DYREQT), built on top of the open-source NASA library OpenMDAO. The tool, DYREQT, formulates an overall problem including a mission description or concept of operations, which maps to the vehicle and its stages through events. According to Edwards et al 2018, their early implementation only used a single-level default MDAO architecture; however, they express the desire for testing other architectures or workflows (as discussed earlier in this chapter) in future development[84]. Not only is the vehicle-and-mission centered MDAO formulation useful for understanding the logistics of the space vehicle, the formulation enables consideration of the vehicle optimization in its trajectory and in a larger system of systems, with multiple vehicles and multiple missions[84]. This approach could be extended to other vehicle classes, like aircraft or rotorcraft, for optimization of fleets of vehicles fit for particular purposes.

4.3 Leveraging MBSE languages for MDO

Some hints have already been developed regarding MDO and MBSE languages. However, there is a very high level of detail in the literature already on this subject. The following sections summarize some of the existing capability.

4.3.1 Product-Based MDO - Geyer's Language: Shape Grammar for Building Optimization

An extremely powerful technique for MDO based on information described in UML and related languages such as SysML is detailed by Geyer 2008[85] and Geyer 2009[86]. Geyer 2008 establishes that a parameter-based description of MDO problems is too limited for exploring different architectural concepts, that the "gap between design practice and the formalism of conventional optimization calls for methods of considering dynamic system modifications in MDO"[85]. In particular, Geyer targets the relation between the MDO problem and a generative description of the system: "the development of shape grammars allows the precise formalization of operations with geometric and spatial elements, which is an important prerequisite for architectural design"[85]. The result is the ability to introduce dramatically altered topology within MDO problems. The mechanism for topological alteration, via shape grammar in UML, is replacements similar to that used by Diagne et al. Specifically, "From the viewpoint of optimization, this principle of replaceability is of major interest since there is usually more than one way of realizing a function. These alternatives open up latitude for improving a design, which is crucial especially during conceptual design"[85]. The grammar which generates these alternatives has a vocabulary and rules — it defines a domain-specific language — and provides for detailing a system decomposition, modifying the system, and repairing the physical description as modifications are encoded. Using the grammar, Geyer assembles a parse tree where the leaf nodes define the disjoint design sets[79], which should be explored via optimization, "such as minimizing costs and environmental impact, or maximizing economic value—leads to optimum designs in terms of parameters"[85]. In particular, a "Design Compiler" is responsible in this formulation for identifying the analytical needs based on the product description[86]. Geyer emphasizes that the traditional approach for MDO relying on a decomposition by discipline is not effective: "Because building components often involve various disciplines, a disciplinary approach would hinder

an easy component exchange by rules and lead to a static, inflexible structure of the optimization model"[85]. In Geyer's UML model, "the components need to contain input and output parameters for linking within the system and analysis procedures for dimensioning"[86]. Interchangeable components accomplish the same functionality, but may do so through alternative technological solutions. The example Geyer uses deals with the structural techniques for a concert hall apparent in the shape grammar parse tree leaf nodes — the use of grillage vs. beams, for example, as "the technologies correspond to different available terminal vocabulary"[86]. Additionally, Geyer recognizes that while his initial effort is similar to MDF, but where the modules in the MDAO graph are replaced by representations of the system architecture instead of disciplines, other MDAO architectures may be useful, such as BLISS, which would require additional development to implement on the basis of the shape grammar approach.

4.3.2 Baseline D1: Application to Aircraft Design

The application of shape grammars has also been applied in the field of aircraft design. Bohnke, Reichwein, and Rudolph established an important criterion for any vehicle-specific design language:

> "The design language therefore needs on the one hand to provide sufficient generic modeling elements to represent engineering data originating from a wide range of disciplines, and on the other hand application-specific modeling elements to recognizably represent the engineering data from the specialized application-specific models."[87]

The specific tool built by Bohnke, Reichwein, and Rudolph is able to map geometry from UML to Excel and CATIA for engineering design applications. Specifically, they use equations from excel to constrain the elements in UML, and use the constrained UML representation to populate a CATIA file. The elements of the vocabulary in-

clude: fuselage, wing, nacelle, flap, intersection, and CFD geometry[87]. UML Activities generate instances according to the UML classes with properties constrained by the imported equations and particular rules for the creation of the elements of the language and the unification of geometric objects. The authors report having created multiple sets of rules, but do not detail them. However, they provide illustrations, claiming to be able to generate conventional and unconventional aircraft designs by modification of the grammar rules. However, despite application to manufacturing simulation[88], there are no examples of this aircraft design language generating MDO architectures as in Geyer. Therefore, Gap 4 is introduced:

Gap 4 Design languages have not been clearly linked to design optimization for aerospace vehicles.

4.3.3 Baseline D2: Evolution to New Design Languages

In his 2015 journal publication corresponding to his thesis, Johannes Gross described a graph-replacement grammar and resulting design language for spacecraft[89, 90]. This graph grammar uses replacement rules to generate system alternatives in the same manner as Geyer[85, 86]. In this case, however, the design language is based in SysML instead of UML. The alternatives generated through the replacement rules are fed once more to the "Design Compiler" for analysis. The leaf nodes of the parse tree have equations embedded in them which permit the computation of system performance for the spacecraft communications subsystem. Specifically, however, "the resulting equation is exported to Mathematica"[89]. This is not unlike the Product-Based Analysis Model (PBAM) techniques discussed in Contribution [61, Paper A] and Contribution [62, Paper B], where equations embedded in constraint blocks are solved symbolically in a tool like Mathematica. While Geyer used optimization directly, Gross focuses instead on solving a constraint graph which results from the system model[90]. With rule formulations which depend in part on requirements,

as the requirements change, the design and thus the constraint graph will evolve accordingly[89]. This enables Gross to explore a design space according to selected performance measures of the system[90], across different sets of design concepts. With the resulting equation system exported to Mathematica per Gross 2015, Gross 2016 describes an additional process of sensitivity testing[90]. This framework is applied to a SysML model of FireSat, a well-known textbook spacecraft with a clear and public description in the famous Space Mission Analysis and Design textbooks (e.g. [91]). While Gross 2016[90] does not specifically address optimization of FireSat, the objective is to generate response surfaces according to parameters of interest for different system solutions according to the design language grammar in an attempt to reduce the cost of exploring the vast combinatorial design space. If Gross's work for product-based, MBSE-centered design space exploration serves as a baseline capability, potentially supported by object-oriented automated optimization tools or advanced constraint solvers external the system model environment, then there yet remains a gap towards answering Research Question 5, noted as Gap 5:

Gap 5 Existing MBSE-based design space exploration languages do not address uncertainty alongside constraint-graph or MDAO analytical capability.

Constraint graphs, elaborated more in constraint theory[92], give rise to acausal networks that be manipulated for search or optimization by formulating a Constraint Satisfaction Problem (CSP) — a technique which has been demonstrate in detail for transformation from SysML for component sizing[93], but which is not fully developed in Gross's work described above. However, Gross does generate a surface describing the performance of the communication system design alternatives in terms of selected design variables, exercising the constraint graph to explore a design space. This is related to the idea of RDS, however, a technique for probabilistic exploration of the solutions sets is also needed.

4.4 Summary of Findings

This chapter has reviewed a large swatch of the MDO literature. Two gaps remain in the state-of-the-art but are particular to the aerospace domain. These gaps include the precise formulation of product-based MDO in terms of an aerospace vehicle representation in UML or SysML (versus constraint graph formulation) from Gap 4, as well as the lack of uncertainty or probabilistics needed for RDS per Gap 5. Therefore, to conclude this chapter, Research Question 6 is introduced to follow on Research Question 5:

Research Question 6 Given a description language for an aerospace vehicle, how should a language for design optimization be interfaced to the description with support for probabilistic analysis?

A plan to address this research question and the previously developed questions will be given in the following chapter.

5

PROPOSED FORMULATION TO BUILD P-SEMP: EXPERIMENTS

> *Practice is the frequent and continued contemplation of the mode of executing any given work, or of the mere operation of the hands, for the conversion of the material in the best and readiest way. Theory is the result of that reasoning which demonstrates and explains that the material wrought has been so converted as to answer the end proposed.*
>
> – Vitruvius, *de Architectura, Book I*

The purpose of this chapter is to clarify the Research Questions resulting from the previous chapters on developing a baseline and investigation of the literature, and to set up a set of hypotheses and experiments which aim to resolve the remaining gaps in understanding. The solution here will be a platform which enhances the capability to perform studies as described thus far, which is called the Platform for Systems Engineering Modeling and Planning or P-SEMP. This chapter aims to summarize the formal argument of this thesis proposal and introduce the experiments, showing all the while that the envisioned problem is the same as that which is to be solved.

5.1 Overview of the Questions

The formal argument of the thesis is based on a series of questions. These questions have been constructed over the previous chapters. They will be summarized here for convenience and clarity.

5.1.1 Initial Questions and Gaps

First, the preliminary questions were established in the introductory chapter. These included:

PRQ 1 How should the SE process be measured?

PRQ 1 is a core objective of this thesis proposal. There are many different measures which can be constructed, but the key ideas align with what measures can be constructed which inform the systems engineer as to the goodness of their approach, but which may also be subjected to simulation for planning purposes.

PRQ 2 How is modeling and simulation applied to the SE process?

PRQ 2 focuses the investigation on the modeling and simulation of aspects of the SE process. That is, for a life cycle process, the aim is to know how the tasks might be simulated, or even for the whole life cycle all at once, how an approach for SE might be evaluated.

PRQ 3 How are decisions made on the content of the SE process?

With PRQ 3, the aim becomes to find answers to measurement and modeling of the SE process such that decisions can be made on tasks for completing the SE process. The result of these three questions is a literature review which constructs an understanding of baseline capability.

In seeking the answers to these preliminary questions, several baseline capabilities were established in the literature. However, the baseline capabilities did not provide a complete solution, due to the following gaps and observations. Firstly, according to Baseline A (Section 2.1), Gap 1 was observed:

Gap 1 COSYSMO models only provide insight into the overall cost of SE for planning purposes, not particular methods.

Gap 1 addresses how COSYSMO evaluates an estimated effort for SE based on attributes of the system and organization. While these attributes may be derived from a system model, the content of the tasks for SE is absent and abstracted through the idea of a level of capability or competency in the SE domain. Subsequently, in Section 2.2 discussing Baseline B, Gap 2 was observed:

Gap 2 Existing fitted models for SE ROI and SE Effort are not applicable to new methods, environments, and paradigms. A forecasting capability which predicts the change in SE measures according to SE tasks is necessary.

For Gap 2, the issue is that with the SE-ROI study, a larger problem has been established in the literature, that both COSYSMO and SE-ROI may be invalid for new ways of doing SE, which is exactly the domain of interest for the preliminary research questions. Instead, a forecasting capability is needed to estimate the relative goodness of planned techniques.

Observation 1 Leading indicators regarding requirements volatility and work product status are highly predictive of review readiness, and are therefore key to measuring cycle-time.

Observation 1 calls out that if the primary interest in seeking out new techniques for SE is to reduce the amount of time between gate reviews, then measures which are more predictive of review-readiness are desirable. Requirements volatility in particular is of interest, since multiple SE processes are involved in the creation and management of requirements. Further, requirements are possibly the most essential and important documentation aspect in SE, especially for the communication of system characteristics among the design team, to suppliers, to contractors, and to regulatory agencies. There are however problems with focusing on the volatility of requirements, as seen in Section 2.4.1 discussing Baseline C1 for a leading indicator model:

Gap 3 How do specific SE methods affect the Walworth et al requirements volatility/status model?

A key issue with models for generating requirements volatility parameter profiles is called out in Gap 3. Gap 3 illustrates how having a model for a leading indicator is not sufficient, as not much insight may be given for the impact of proposed SE methodology in terms of the SE methods/tasks. In the case of Walworth et al[42], that is the point of their work, that "the SD model developed in [their] work is not a functionalist representation of the project as a system". Other techniques may still be needed. However, Baselines C2 and C3 in Sections 2.4.2 and 2.4.3 were found to be lacking:

Observation 2 CERs fitted to SE practice are inherently SE methodology-independent. Deeper analysis is necessary to model the effects of particular SE methodology.

Observation 2 calls out another issue with the incremental improvements to COSYSMO and other CER-type models, in that in order to have sufficient data for fitting their models, these tools require an aggregation of data from SE methods. As more data is aggregated, the models may provide a better picture of the SE field as a whole; however, due to aggregation, they may never have the granularity needed to study individual tasks and will always be limited by the historical nature of their data when applied to new SE practice.

The next two gaps go together, drilling into the details for integrated analytical capability for MBSE languages. These gaps were developed in Sections 4.3.2 and 4.3.3 for Baselines D1 and D2 respectively.

Gap 4 Design languages have not been clearly linked to design optimization for aerospace vehicles.

Gap 4 states that while design languages have seen clear application to MDO for buildings, they have not be applied to the traditional disciplines in aerospace vehicle design for a product-based MDO formulation outside some manufacturing-focused applications.

Gap 5 Existing MBSE-based design space exploration languages do not address uncertainty alongside constraint-graph or MDAO analytical capability.

Gap 5 states that there is a gap in the expression of design variability and design analysis for MBSE languages, specifically in the application of uncertainty in these formulations. Uncertainty and probabilistic techniques are necessary for RDS and early validation of systems.

5.1.2 Summary of Research Questions

In order to determine a means to close these gaps, an additional literature review was pursued for new modeling techniques. The following research questions attempt to direct the investigation towards a solution which cross-fertilizes ideas from different domains to assist in analyzing the impact of SE tasks towards process outcomes.

Research Question 1 How can any particular SE methodology be shown to be measurably better than another?

Research Question 1 is a restatement of PRQ 1 and PRQ 3, as this is truly the larger question that ought to be answered. If a technique can be used to discriminate between proposals for SE methodologies for a given context of PMTE, then the various businesses and organizations engaged in SE can improve their decisions, cut costs, cut time to market, and build better systems. However, SE is very broad in scope, and for this reason, some actions will need to be taken to narrow down the problem.

Research Question 2 How can simulation of SE processes be tied to leading indicator models for planning and project management?

Research Question 2 combines PRQ 2 and PRQ 3 based on Observation 1 and Gap 3 to focus the effort towards answering Research Question 1. Specifically, the intent will be to seek out a means of applying models of leading indicators towards planning SE activities.

Observation 3 Multi-level business process modeling could be used to gain insight on how to structure lower-level process in a SE "shop" on the high-level leading indicators.

Observation 3 points out that the literature for business-process re-engineering provides analytical techniques for studying the impact of business tasks on measures of performance, and that many of these same techniques might be applied to tasks for SE.

Research Question 3 How should the SE methodology be represented so that it can be simulated?

Building on Observation 3 and adding more detail underneath Research Question 2, Research Question 3 specifically seeks a means of representing the SE tasks in such a way that they can be simulated, so that measures like leading indicators might be calculated from the proposed task simulation. From Research Question 3, Research Question 4 was derived:

Research Question 4 How should a transformation be constructed for simulation of SE methods via DES?

Research Question 4 assumes that the representation of the SE tasks resides in a semi-formal language such as SysML. From this representation, a translation must be constructed such that the resulting model is amenable to simulation. Interesting questions at this point arise for closer inspection during experimentation, especially regarding the pros and cons of simulation models like DES, and whether building

a hybrid model according to Observation 3 using the material discussed in Gap 3 constructed from the soft-systems perspective is at all compatible with models such as DES.

Research Question 5 How should a DSL for RDS activities interface to M&S, uncertainty, and other infrastructure from a system model?

Finally, drilling into the tasks for the validation process, especially early-phase validation of requirements, there may be a need to better-incorporate and leverage existing design methodology within the SE ecosystem. Research Question 5 asks how this embedding might be formulated so that these design methodologies are more accessible when performing the SE tasks for validation in early phases. Going one step further, Research Question 6 asks specifically, given the existence of aerospace vehicle design languages embedded in MBSE langauges, how design optimization and probabilistic analysis might be better represented.

Research Question 6 Given a description language for an aerospace vehicle, how should a language for design optimization be interfaced to the description with support for probabilistic analysis?

5.2 Arguing for the Questions: Formulation of Hypotheses

In order to begin building a platform capable of addressing the research questions, several hypotheses must be established. Research Question 1 seeks to find a means of selecting among SE methods according to measures. A broad answer to Research Question 1 is to apply models which must be parameterized in terms of the SE tasks and which calculate criteria like a requirements volatility leading indicator for use in decision-making. Hypothesis 1 states this concept in terms of the idea behind leading indicators; the absolute criteria for SE, as a business process, must be related to cost, schedule, success — possibly even profit[24].

Hypothesis 1 If a model for a SE measure predicts progress in a SE process in correlation with a lagging indicator such as cost, schedule (often combined as effort), etc, then the model provides a basis for decision making on the method for a SE process.

However, these lagging indicators may not be sufficient alone in planning for SE or in taking action in the form of course-corrections. Despite criticisms which have been leveraged towards the Walworth et al 2016 model in this document, that model could be used to decide what size team ought to be put on the task of managing the requirements. The s-curves produced by the model represent trends towards gate-review readiness. Accelerating the pace of readiness brings the system to completion faster. Thus, a trade between business resources, time to market, and presumably system success may be possible.

Such a sweeping exploration is far outside the scope of a single thesis. Therefore, the lens of this investigation must be narrowed down to one portion of SE. The specific process of interest will be validation, especially validation actions which occur early in the system life cycle in the attempt to establish feasibility. Hypothesis 1a extends Hypothesis 1 and seeks to find models according to methods which modify requirements, such as the Grady Validation Process in Figure 3.2.

Hypothesis 1a If requirements status is parameterized, then important parameters can be identified and understood.

The Walworth et al 2016 model presents one possible parameterization of requirements status according to the learning system metaphor. However, the question remains as to whether more insight might be provided in terms of the SE tasks to be performed. Thus there are two additional hypotheses regarding the components of what might be a platform for comparing SE methodology.

First of all, it could be that other simulation paradigms are better for extracting the information necessary for these particular decisions. Addressing aspects of Research Question 2 in terms of Observation 3, as well as Research Question 3.

Hypothesis 2 If SE methods are also processes per Martin[4], then they can be simulated by process models like DES or ABS.

While Hypothesis 2 may not be falsifiable, its success may result in questions regarding the utility of the result. The functionalist critique of a "hard-system" from the soft-systems perspective as in Walworth et al 2016[42] is the most likely source. However, given such a simulation model as DES or ABS, then the SE tasking is now an integral part of the simulation. If the results of this simulation are linked to the computation of measures like leading indicators for requirements volatility, then the objective is complete — changes to the task plan result in new evaluations of the leading indicator trajectory. This goal could be further enhanced by Hypothesis 3:

Hypothesis 3 If SE methods are treated like other business processes, then bi-level hybrid simulation will provide better parameterization for task-planning purposes.

Hypothesis 3 states that a bi-level hybrid simulation is needed. There is a risk of course that the Walworth et al 2016 model is not compatible as-is with bi-level simulation; either 1) a new SD model would be needed, or 2) the bi-level paradigm is not appropriate for this case. In the case of 2) the hypothesis is rejected, however whichever model is formulated alongside the SD model should essentially calculate similar trajectories yet do so according to the task plan. This resulting model can still accomplish the other objectives of this study. Additionally, comparison with the SD model may be even more useful than combination; for example, Brooks[26] claims that adding more engineers to a late development effort will make it later.

Whether this negative feedback is foreseen by the SD model from Walworth et al as-is, probably not; adding more engineers should always increase the rate at which the work is completed. New modifications to Walworth — perhaps a recursive learning model — or new models applying DES might be able to model such effects. In either case, situations as described by Brooks are precisely what decision-makers face for SE methodological changes during product development.

Hypothesis 4 is established with respect to Research Question 5 and Research Question 6.

Hypothesis 4 If a labeling for probabilistic methods is applied to a customization of an MBSE language with transformation via intermediary representation to target analytical environments, then confidence will be increased in the system representation of the resulting analytical models.

According to the literature[64], it is already known that the technology behind realizing Hypothesis 4 should feasible. Additionally, similar labelings and transformations to intermediate representations have been previously implemented by this author for design optimization from SysML for aircraft and spacecraft domains. The real issue which must be addressed is the disjoint nature of the system model and design optimization vs the RDS process, to ensure that steps for performing validation are conducted using valid analytical models.

5.2.1 A Note Regarding the Correspondence of the Hypotheses to the Problem

A pitfall to be avoided is whether the problem which is constructed and presumable to be solved here corresponds to the real problem which is intended to be solved. Thus far, no experiments have been elaborated, and with the experiments is where the risk may be greatest. However, should the first and second hypotheses be true, then at least for validation actions in early phases, the real problem will be addressed. By constructing simulation models as a function of the task plan for a leading indicator,

which is known to correlate to cost and schedule, decision-making on the content of the task plan can be enriched at a quantitative level beyond qualitative characteristics of PMTE options for SE methodology. The critical aspect now is the specification and later completion of the experiments.

5.3 Development of Experiments

Several experiments are planned regarding the hypotheses presented above. These experiments seek to establish the missing capability identified with respect to the literature. In order to determine how to best perform SE, *the key objective of this dissertation*, it will be necessary to compare different methods for identical processes tailored to a given system and organization.

5.3.1 Experiment 1: Replication of the System Dynamics Model

Experiment 1 Replication of the Walworth et al System Dynamics Model.

Rationale The first step in building the platform will be to replicate the system dynamics model from Walworth et al[42] and immediately satisfy Hypothesis 1a. The system dynamics model from Walworth et al is a "learning system" archetype and therefore provides insight to both the requirements status, and the organizational understanding of the requirements. While a major part of requirements analysis in terms of requirements added, modified, or deleted, this type of model also represents aspects of requirements validation practices. Thus, it lays a foundation for further investigation. The technique will be to recreate the System Dynamics model using the OpenModelica SystemDynamics library[94], and replicate the behavior illustrated in the paper to recover the learning-system analogy to requirements status. However, as far as tools, it may turn out that the open-source library is not reliably functional. In this

case, other tools in javascript or python my be necessary, including a custom simulator. Regardless of the precise tooling, this experiment is summarized by the statement in Experiment 1.

Purpose To provide a canonical model for requirements volatility that can be tightly integrated with a system model representation.

Data Required The paper by Walworth et al 2016

Models Required A replica of the System Dynamics model which is capable of similar parameter variation to Walworth et al 2016

Experimental Procedure :

- Construct System Dynamics Model in modeling tool
- Vary model parameters to obtain s-curves from literature
- Identify possible parameters to be calculated by method models
- Explore integration with a system model

Expected Outcome A quantitative means of exploring the impact of SE practice on validation tasks which has already been vetted in the literature. This model will support future experiments in the creation of hybrid models which provide greater insight towards the impacts of tasks on requirements volatility, or alternatively as a comparison against these other models.

5.3.2 Experiment 2: Development of Method Models

Experiment 2 Formulation and testing of DES models representing a validation process

Rationale The experimental procedure in response to Hypothesis 2 and Hypothesis 3 is to formulate simulation models firstly for known validation methods, such as

Grady's textbook diagram from Figure 3.2. These flowchart descriptions might normally be simulated by DES. The first step will be to construct a model of each kind. Then, the model must be run to examine parameter variation. There will be variation of the distribution parameters of the simulation models first to see if the behavior and performance of the models is similar. Then, having established a model, the effect of the graph structure on the simulation model should be noted for later study. This procedure is summarized by the statement in Experiment 2.

Purpose To create a capability for simulation models of SE tasks, i.e. the methods for accomplishing SE processes. This will be specifically applied to tasks for validation in early-phase development.

Data Required Required data includes validation methods as found in the literature, according to a task flow. Information on the rates of task completion would help to calibrate and validation the task models; however, data on task completion rates is unlikely to be available.

Models Required Models of validation methods will be constructed in appropriate modeling tools according to the data from the literature.

Experimental Procedure :

- Construct DES (e.g. PyDES)
- Simulate DES with some parameter variation to see its impact
- Check for compatibility regarding the System Dynamics model
- Plan a mapping of the DES model to a system model

Expected Outcome The expected outcome of Experiment 2 are several DES models representing validation methods.

5.3.3 Experiment 3: Integration of Models

Experiment 3a

Experiment 3a Investigation of hybrid simulation for DES and SD models

Rationale The third experiment has two parts. After formulating a small number of method models in Experiment 2 and testing them for their behavior independently, the method models must be interfaced to the System Dynamics model from Experiment 1. Experiment 3a corresponds to Hypothesis 3. If Experiment 3a succeeds, then the hybrid model will provide requirements volatility trends as a function of SE tasks. However, should it prove impossible to combine the method simulation model with the SD model in a hybrid simulation, these models are still useful individually for the overall purposes of this investigation as discussed above.

Purpose A hybrid simulation model may provide better qualitative benefits to decision-makers by giving different levels of abstraction for business operations, and if this experiment succeeds, such a model will be produced. However, if the experiment fails and the method models are instead an alternative to the System Dynamics model, then they can be compared.

Data Required Models constructed during Experiment 1 and Experiment 2.

Models Required Models previously built.

Experimental Procedure :

- If the models built thus far are compatible, construct an appropriate software interface for co-simulation
- If the models are incompatible, provide a comparison of the calculation of requirements volatility in terms of validation tasks vs. the Walworth et al characterization of the organization as a learning system.

Expected Outcome An integrated simulation model or comparison data between the soft-systems-based model and a hard-systems-based approach.

Experiment 3b

Experiment 3b Generate the method models and/or combined simulation from SysML

Rationale The second part of Experiment 3 relates to representation. That is, how to manage the variation of the individual and/or combined models. While some variation might be based solely on parameters, DES models of tasks have a high combinatorial degree of freedom when considering possible modifications of the directed graph. Additionally, as with Sprock and McGinnis 2015[65], it is desirable for the planning of the SE tasks to be tightly integrated with the SE model or documentation (as discussed by Martin for the document-based case[4]). To enable these capabilities, the simulation environment should be created according to a system model description, per Experiment 3b. The choice of SE language is only important insofar as the tools and environment of PMTE are convenient, as is the case of SysML for this author. While the specific transformation will be different in implementation from other languages, or possibly even the same language in different tools and environments, this author has developed several capabilities in an existing set of tools which may prove useful for building Experiment 3b.

Purpose Enable generation of alternatives which are recorded in the system model as part of the SE management plan.

Data Required Models previously built.

Models Required A system model capable of recording the simulation model data with appropriate parsing to extract simulation models.

Experimental Procedure :

- Develop a system model
- Create simple simulations of the system model, perhaps using a tool such as Cameo Simulation Toolkit
- Specify stricter transformation rules to the simulation tools used in previous experiments
- Extract simulation models according to transformation rules

Expected Outcome A system model and simulation capability which enables an MBSE-based exploration of the SE task planning exercise and use of the leading-indicator models.

5.3.4 Experiment 4: Exploration of Model Parameters

Experiment 4 Explore the sensitivity of the leading indicator to changes in the method model.

Rationale Experiment 4 seeks to address Hypothesis 1 with the completed environment. Given the capabilities built over the previous experiments, it is now possible to investigate how the leading indicator responds to the task graph and to understand the impact of planning choices. With Experiment 4, the full capability to explore method proposals would be available. Future work could attempt to provide concrete correlations between the leading indicator and cost models, a missing functional relationship that impedes getting costing numbers in terms of the SE tasks. The platform enables the consideration of new SE methodology proposals in terms of their impact on a leading indicator trajectory, i.e. an s-curve, at least in the case of requirements volatility during early-phase validation actions.

Purpose To provide a characterization of the final hybrid or independent models in terms of a task plan. This effort could be used to support optimization of the task plan, decision-making on the content of the plan, or future techniques for SE cost analysis.

Data Required Models built thus far

Models Required New models for the responses (i.e. leading indicator measure) in terms of variables which parameterize the task plan.

Experimental Procedure :

- Parameterize the system model representation
- Use the system model representation to generate simulation models
- Evaluate the simulation models over a design of experiments
- Fit surrogate models to the leading indicator measure
- Use the surrogate models of the leading indicator to explore the SE method design space

Expected Outcome Visual representation of the trades available from the SE method alternatives

5.3.5 Summary of the First Portion

Figure 5.1 illustrates the relationship of Experiment 1, Experiment 2, Experiment 3a, Experiment 3b, and Experiment 4.

5.4 Addressing Design Capability in the SE Process

Having established a platform for the comparison of SE methodology, it may also be of interest to take a deep-dive into the method(s) under consideration. Additionally,

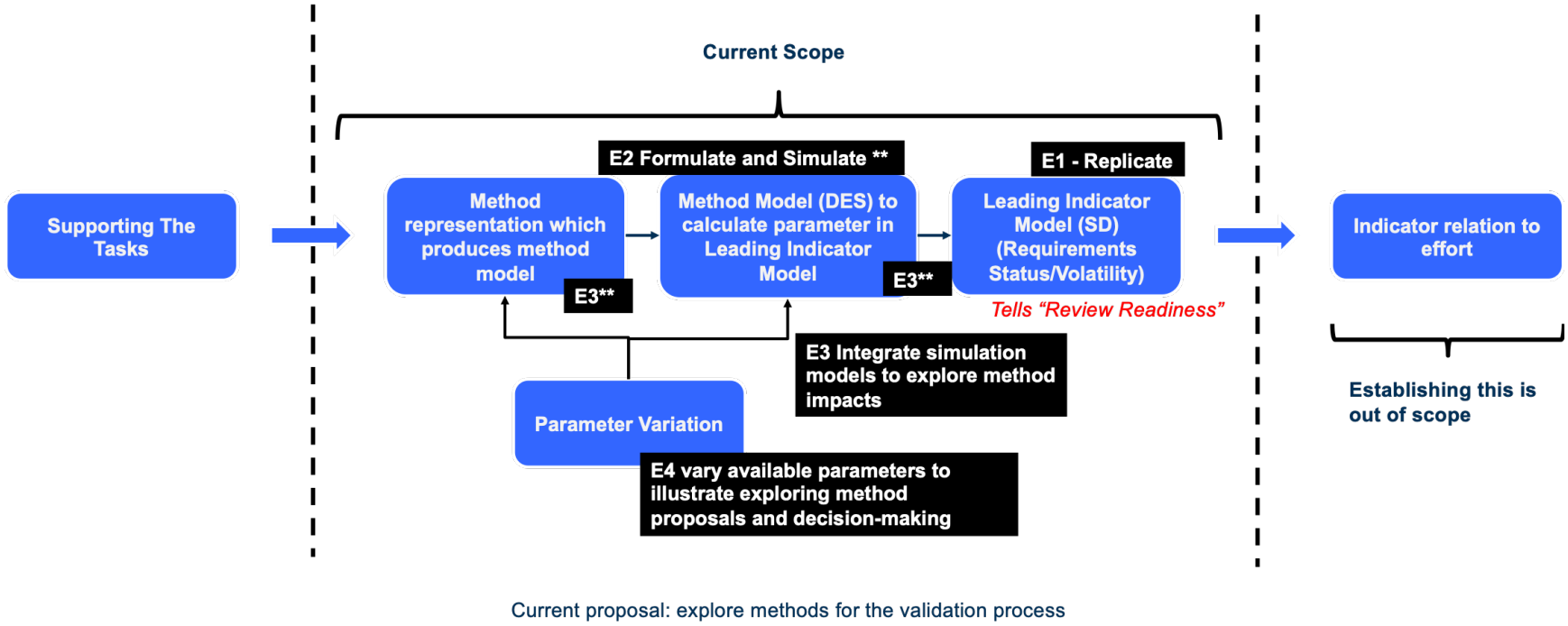


Figure 5.1: Layout of the experiments for SE method simulation

uncertainty must be accounted for within the considering of the validation actions. A potentially suitable validation method during early-phases, interested in the quantification of requirements according to feasible values, might be RDS. In the case of RDS, it is desirable not only that the values returned populate valid requirements, but that the setup for RDS is based on a valid description of the system of interest. Thus the RDS capability must be tightly integrated with the system model, such that the coordination of surrogate model creation from M&S or MDAO is appropriate, and that the application of Monte Carlo methods is tracked for proper recording keeping as to the bounds for the variables, which are a sort of requirement. The result may be a DSL for RDS activities in the system model.

5.4.1 Experiment 5: Improvements for MDO and RDS Representation in MBSE

Experiment 5 Executable expression of vehicle design and RDS in an MBSE Language

Rationale Tight integration of modeling and simulation to the MBSE language helps to support system validation as well as model validation. These actions are especially important in the early phases of the life cycle. Ensuring that the model representation embedded within an analysis is an important part of this integration and answering Hypothesis 4, and the formal techniques by which this can be done have been established for many years. Implementation-wise, it will be similar to generating the simulation of the SE process from its description in SysML; however, the analytical model produced will be for some optimization problem either in the context of MDO and/or RDS. These sorts of transformations have been discussed in detail in the preceding chapters and continue to be a subject of research (e.g. [95]).

Purpose Transform vehicle description in the MBSE language to RDS and MDO problem(s)

Data Required :

- Vehicle description
- Vehicle performance relations
- RDS, MDO platform ready for interfacing

Models Required A SysML model of a vehicle and various analytical models corresponding to evaluating its performance

Experimental Procedure This technique has been used twice before by the author for incorporating MDO with MBSE languages, first in 2017 for a space vehicle problem and again in 2019 for an aircraft problem. The following must be constructed:

$$\Gamma_{S_v} \underset{L}{\mapsto} \Gamma_{I_v}$$

$$\Gamma_{I_v} \underset{M}{\longmapsto} \Gamma_{E_v}$$

In this case, Γ_{S_v} is some subset of a SysML model of the vehicle with a particular labeling, Γ_{I_v} is an intermediate representation of the vehicle content, and Γ_{E_v} is the external representation of the vehicle. This is close to the approach advocated by Cole and Dinkel [64], however it does not require that all the content of Γ_{S_v} is translated to Γ_{I_v}, nor does it require that Γ_{I_v} is in serialized form. Usually, Γ_{I_v} is a sort of list or associative array containing the desired information in memory. This is a less-formal transformation, where customization happens both at the level of Γ_{S_v} being a near template of the information needed by Γ_{E_v}, as well as within the sets of rules L and M. The rules L and M describe the translation between the model representations; in effect they form the model-transformation. In past experience, Γ_{E_v} is in JSON format to be ingested by the analytical program. The analytical program configures its

analytical models as far as it is capable of in terms of Γ_{E_v}. One option which may be useful in particular for the probabilistic aspects of RDS may prove to be probabilistic programming languages, which have seen much recent development[96, 97]. These languages are reportedly in the declarative programming paradigm, aligning with techniques discussed in Contribution [61, Paper A].

Expected Outcome A capability to run vehicle design problems through RDS techniques based on the content of the system model.

5.5 Use Case - Alternative SE Processes and Design of FireSat

Friedenthal and Oster[98] recently laid out a detailed MBSE approach for spacecraft, especially regarding early-phase spacecraft design of the like described in the New SMAD textbook[91]. In agreement with Contribution [62, Paper B], Friedenthal and Oster state that:

> "[system analysis] is performed concurrently with the Specify System Requirements activity to specify the critical system black box performance, physical, and other design characteristics. The required property values and the estimated design values are updated based on iterative analysis of the mission design and the system design"[98, p. 62].

The procedures in Friedenthal and Oster's book are a simplified representation of OOSEM with an publicly-available SysML model. The steps of specifying requirements and synthesize alternative architectures involve repeated application of aspects of early-phase validation, where requirements are being added, modified, or deleted as they are quantified and as the system architecture is revised. Further, this example meshes well with advanced system design languages such as by Gross[89, 90]. This presents a good launching point for enhancing a baseline system model, with a baseline systems engineering process, and for creating an environment like Edwards et

al[84] based on design relations available in the open literature[91] for a well-known canonical system of interest. Past experience[99] running the SMAD[91] relations through a PIDO environment interfaced with a system model (technology generally described in Contribution [61, Paper A]) further support this choice as efficient and effective.

5.6 Summary

Five experiments are proposed for this thesis. P-SEMP is mainly built by the first four. The validity of the results of P-SEMP, especially when the validation process uses RDS, may be enhanced by Experiment 5. Overall, the development needs a use case and plan, which are briefly detailed in the subsequent chapter.

6

SUMMARY OF THE P-SEMP METHODOLOGY

The Platform for Systems Engineering Modeling and Planning (P-SEMP) can be considered as a methodology in terms of constructing the platform, and exercising the platform. Specifically, the platform built in the P-SEMP thesis following the associated P-SEMP methodology will focus on tasks and task planning in Systems Engineering Methodology (SEM). The P-SEMP methodology informs the implementation and use of the experiments defined according to the argument of posed by the P-SEMP thesis and requires further elaboration.

6.1 Discussion on Underlying Philosophy

Some of the views of the P-SEMP methodology are similar to Gilbertson's 2018 Systems Engineering Method Diagnostic Assessment Model (SEMDAM), while other aspects differ. According to Gilbertson, the main focus of SEM diagnostics is to decide between one of the following: Traditional Systems Method, System-of-Systems Method, Enterprise Systems Method, and Complex Systems Method, or No Systems Engineering Method[100, pp. 85–88]. It is this structure that is the main point of disagreement in fact between P-SEMP and SEMDAM at a high level — that is, for traditional systems engineering and even model-based systems engineering, there exists more than one methodology (e.g. as described by Martin, Estefan[1], and others). Thus the level of detail proposed by Gilbertson is much broader, much less fine-grained, and covers entire swaths of potential task definition in what are effectively methodological categories. This is important, as for the domain of SEM, there is currently an environment in which practitioners must decide whether to effectively buy-in to a particular SEM from a tool or environment vendor. For example, Gilbert-

son's SEMDAM would not necessarily assist in a decision as to whether a business ought to purchase licenses for MagicGrid[101], which is being sold as a SE methodology to subsume all SE methodologies by the authors Morkevicius et al 2020[101], or to follow a version of RFLP by Dassault[102], or Arcadia[103] when at the same level of system complexity. It is this issue, the concrete modeling of the pros and cons of specific tasks using the tools and environment of systems engineering which are so often sold on the basis of presentation and marketing or likewise on the experience of a senior systems architect, which defines P-SEMP and its objectives. That is, to take the task proposals one might provide for SEM literally and try to understand their impact on the bottom line. In Gilbertson's terms, P-SEMP is analysis rather than assessment offering decisions rather than diagnostics[100, p. 112]. This is due to the "detailed examination"[100] of the complex task architectures that is crucial to P-SEMP and specifically the modeling activity.

However, there is a key point of agreement between P-SEMP and SEMDAM. That point of agreement is in the application of abductive reasoning. That is, according to Douven's "ABD2", that "given evidence E and candidate explanations $H_1 ... H_n$ of E, infer the truth of that H_i which explains E best, provided H_i is satisfactory/good enough [as] explanation"[104]. The implication taken here for P-SEMP is to permit a cycle of assumptions as H_i providing explanations for evidence, such that one hypothesis may provide evidence for a subsequent in a chain of assumptions. In effect, this approach is seen as crucial to model-building or prototyping. Consider, first systems engineers discover what work needs to be done for requirements engineering tasks. Then, assume that the Walworth et al model calculates the learning rate behavior by which the Systems Engineers discover work. In this case, the truth of the statement might be taken on the basis of a journal paper, or by some other means, as a mechanism for establishing a model that provide insight towards some further objective, such as when the Systems Engineers might finish the requirements engineering

tasks. Each experiment in this thesis follows a pattern of prototype development and model-building, insofar as assumptions are made as to what modeling will work and prototype models are constructed; occasionally, they will fail, resulting in information whether the model-building succeeds and achieves the intended objective, or if it fails for unforeseen reasons delivering new lessons. This benefit of the prototype development cycle, a practical aspect of abductive reasoning in the real-world sense, where an engineer develops knowledge on the basis of what works and what does not, is the core element behind P-SEMP and how P-SEMP proposes to avoid dogma over one method or another, but to take each in turn and model them as is suitable and measure performance to monitor progress.

6.2 Comparison of Methodologies for Selection of Systems Engineering Methodology

There may be some existing methodologies for selecting Systems Engineering methodology (aka SEM Selection Methodologies). The intent here is to discuss in greater detail the existing methodologies referred to above; specifically regarding SEMDAM [100] and Morkevicius et al[101], but also Honour [2]. Due to its simplicity, Morkevicius is considered first.

6.2.1 Ad-Hoc Methodology for Selection

Specifically, Figure 6.1 extracts the comparisons provided by Morkevicius et al 2020[101]. These comparisons represent a recent effort, published during the duration of this thesis, which compared Systems Engineering methodologies for the purpose of selection and/or proposing some improved SE methodology. However, there are many issues with the comparison. First of all, their proposed SEM Selection Methodology has no information given in the form of the first three, which are described with strengths and weaknesses. The first three SE methodologies are summarized with strengths and

weaknesses, and then an architecture framework — noting that per ISO 42010[13] that architecture framework is not equivalent to Systems Engineering methodology but one potential aspect of Martin's [4] PMTE, specifically a tool or perhaps environment — leaving only a concluding statement that such description of the architecture framework proves the efficacy of the proposed Systems Engineering methodology. Further complicating matters, the authors are all from the same company, and three of the SE methodologies compared, including the one they propose, are supported, sold, or otherwise products of their company, while the SE methodology panned by their comparison is provided by a market competitor. No numerical analysis, performance analysis on the basis of SE measurement, discussion of cost or schedule overrun, or project success is provided by the authors as motivation for considering other SE methodologies. The primary concern appears to be support for specific views and software functionality within a software environment for authoring SysML models, presented across subjective comparison statements. Importantly, the assertion of this thesis is that such shortcomings are not entirely the fault of the authors in [101], but that this approach represents the baseline SEM Selection Methodology in common practice for selection of Systems Engineering methodology. That is, the baseline SEM Selection Methodology enjoying widespread use today is an ad-hoc approach which relies entirely on subjective criteria, tool compatibility, and does not focus on task performance, task improvement, or otherwise quantitative assessment of the efficacy of one method versus another for a project, organization, program, vehicle, or team. While at first such approaches may seem acceptable to practitioners as a pragmatic means to decide what software environments they should purchase, the lack of basis in measurement hampers further process improvement through quantitative analysis, as whole portions of PMTE are substituted without any targeted measures implemented, tracked, and modeled. However, P-SEMP is not the first effort to provide better systematization of the selection methodology.

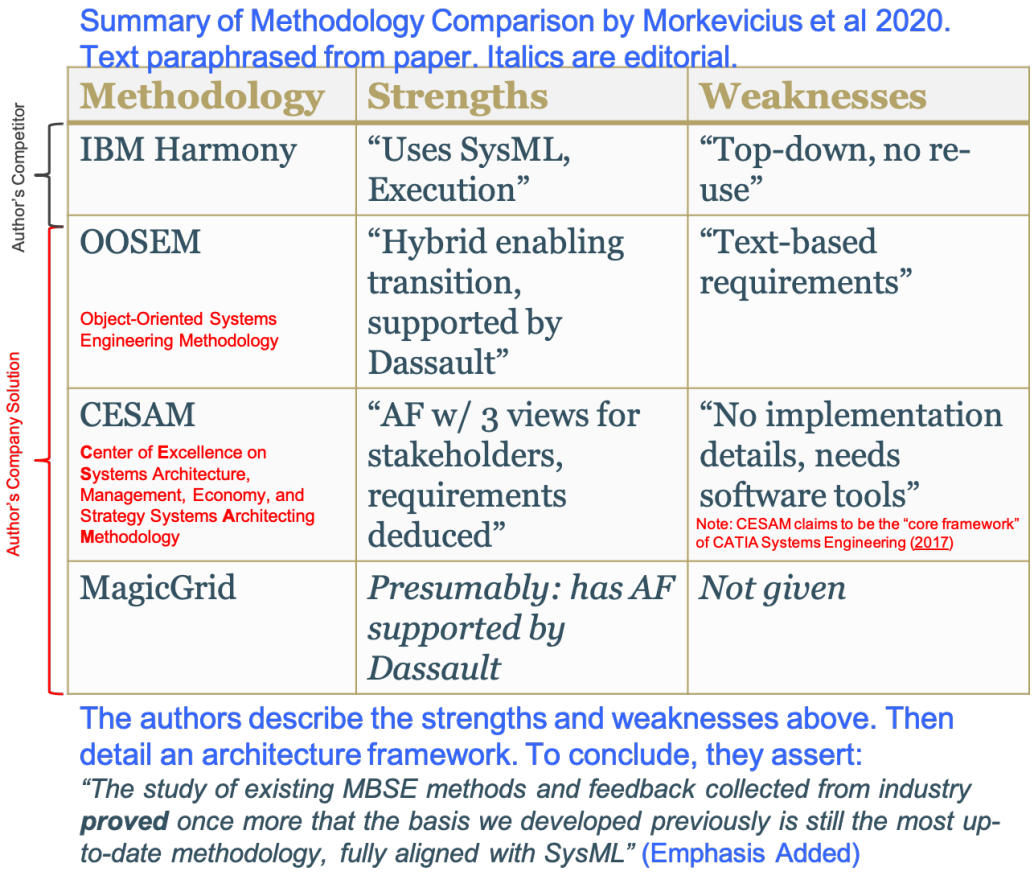


Figure 6.1: Extracting the Comparisons in [101]

6.2.2 Systematic Methodology for Categorization

As discussed above, Gilbertson's SEMDAM [100] sought to provide an analytical, diagnostic capability to aid in selection of a Systems Engineering approach in the academic literature. Strictly speaking, the categories investigated by Gilbertson do not map to SE methodology in PMTE as defined by Martin [4] and Estefan[1]. These categories were Traditional Systems Method, System-of-Systems Method, Enterprise Systems Method, and Complex Systems Method, or No Systems Engineering Method[100, pp. 85–88]. Specifically, SEMDAM determines the "class of system problem" (COSP)[100, p. 4], done "based on evidence and logic not characteristics and assumptions" [100, p. 7], and relying on the Cynefin technique. Note that the output described as a method or methodology in SEMDAM, listed above, is actually

a categorization of SE methodologies defined by PMTE, so that for each classification based on COSP there would be some set of appropriate PMTE available to choose from to perform SE tasks; the precise SE methodology in the sense described by the Morkevicius comparison is yet to be determined. SEMDAM recommends a series of procedures to determine the method categorization. These procedures are illustrated with respect to healthcare in the United States. First, SEMDAM gathers evidence regarding what activity exists for SE and management, finding that some efforts to classify healthcare would label it as a system of systems and confirms that "evidence of previous SE&M activity exits"[100, p. 145]. Given this information, in the second step SEDAM evaluates a first abductive hypothesis and finds that healthcare is not a disordered COSP. In the third step, the stakeholder needs for healthcare as a system is determined from the literature. in the fourth step, business goals are defined per literature to "Deploy Health Information Technology (HIT) that protects privacy and enables data integration"[100, p. 158]. In the fifth step, based on an assumption of a known COSP, SEMDAM begins to "identify potential attributes and associated statistical models"[100, p. 158]. In the demonstration, the result is that the prediction factor is the categorical variable of breach kind, the response is the continuous variable individuals affected, and the statistical model is a two-sample t-test. The findings specifically in this fifth step were that the assumptions about the solution procedure, increase automation, did not necessarily correlate to a reduction in breaches. The sixth step evaluated whether the assumed known COSP was still valid, and found that it was not and instead reduced the statement to knowable COSP. At this point, SEMDAM begins to iterate. Further statistical tests are formulated and performed, modifying the COSP if appropriate based on inference from the statistical tests. Returning to step 5 but for knowable COSP, different data is collected to formulate a new statistical model and perform a hypothesis test. In step 6 again, the results of the test lead to migrating the COSP for healthcare to complex. Step 5 iterates again

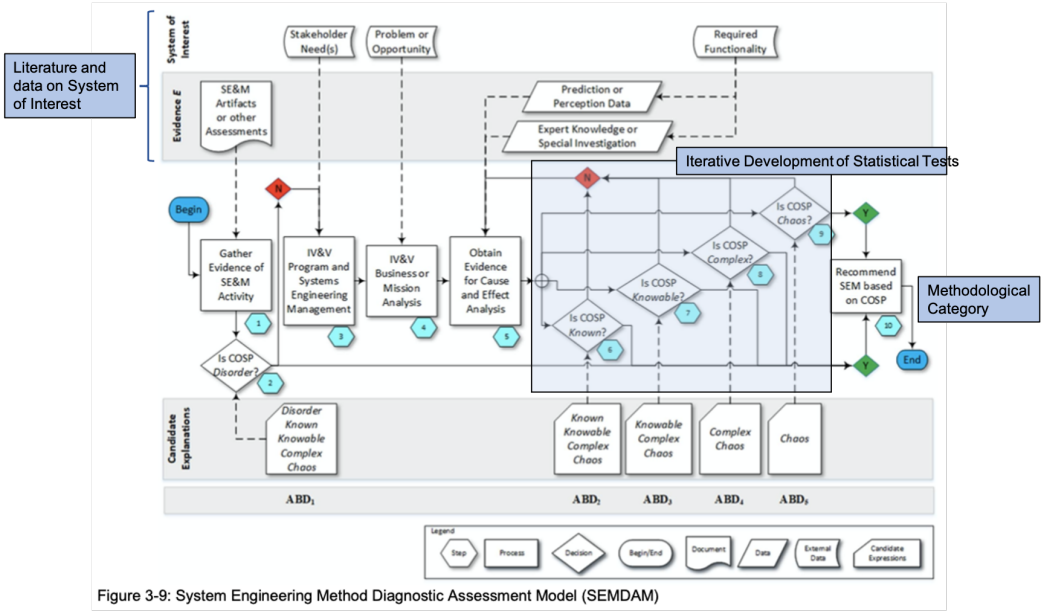


Figure 3-9: System Engineering Method Diagnostic Assessment Model (SEMDAM)

Figure 6.2: This figure show's Gilberton's [100, p. 114] methodological illustration, with additional markup to highlight the research and data collection, iterative development of statistical tests, and output of methodological categorization

with an updated model with different attributes. Again the model is similar in structure as a two-sample t-test. As a result, in the return to step 6, SEMDAM asserts that the COSP for healthcare is complex[100, p. 172]. There are a few useful aspects of SEMDAM to consider, the process of which is illustrated in Figure 6.2, especially in contrast to Morkevicius et al. Note that in each of the steps involving the status change for COSP, a concrete statistical model is formulated on the basis of measurable variables related to the system of interest. The categorization is determined on the basis of the results of hypothesis tests of these statistical models. The resulting summary statistics, parameters, and data remains to help inform stakeholders as to the factors which provide the greatest influence towards their domain. This is a desirable feature of SEMDAM and its approach focused on measurement, similar to P-SEMP. However, some issues remain. For example, as a diagnostic tool, SEMDAM does not provide detail to contest or affirm the claims made by Morkevicius, which are asserted here as typical of claims seen in selection of Systems Engineering methodology. This

is because in terms of any systems engineering approach, for example focused on one particular standard or another, there are often multiple techniques or tasks available to complete the specified process. For example, while SEMDAM focuses on MITRE [100, e.g. p. 100, referred to 32 times] and IEEE systems engineering literature, and also mentions ISO 15288, it is important to note that the MITRE literature provides greater specificity compared to 15288, approaching the kinds of architecture framework view specification seen in Morkevicius et al, whereas 15288 and IEEE standards remain at the higher level described by Martin's definition of process. Thus, even if the COSP from SEMDAM indicates a particular categorization of SE methodologies, it remains to be seen which tasks and in what sequence will be determined for the Systems Engineering Management Plan and related scheduling documentation.

6.2.3 Resource Allocation and Sizing

Returning to the SE-ROI studies discussed in Baseline B, Honour[2], a reasonable concern is whether SE planners could just use historical data and fitted models to determine the appropriate allocation of effort on the SE process elements which they will pursue. However, as mentioned in the earlier discussion of SE-ROI, methodological differences were confounded in the survey data. That is, the SE-ROI model lumps together all SE methodologies and may not include much information under the MBSE paradigm, nor industries outside of the typical aerospace actors in the defense, aeronautics, and astronautics fields. As such the Baseline B discussion concluded that both SE-ROI and COSYSMO would be inadequate as a predictive capability for new methods targeting the standardized SE processes, and that some other predictive capability would be necessary. However, perhaps a team will implement Structured Analysis or Grady's own take on MBSE[35] and assume that the data is good enough. In that case, a tool like SE-ROI could potentially be leveraged to help analyze the resource allocation (effort usually being person-hours, and sometimes also combined

with cost). Notably, SE-ROI formulates its models in terms of the investment needed to meet budget, meet schedule, and achieve project success, so if these attributes are of primary importance, the technique could gain additional favor given appropriate methodological boundaries. Unfortunately, if the management team is trying to analyze and compare works such as Morkevicius[101] and see what the true impact would be for them if they acquired the product of those authors, SE-ROI would not be able to assist due to the content being outside the data set entirely, being focused on an MBSE-based Architecture Framework definition, and generally focused on the method/tool aspect of SE methodology rather than the SE lifecycle processes.

6.3 The Proposed Methodology of P-SEMP

Having established the philosophical basis of P-SEMP, and compared existing SEM Selection Methodologies, the proposed SEM Selection Methodology itself consists of two phases: constructing the platform and exercising the platform, illustrated in Figure 6.3.

This generic visualization will be examined further shortly on the basis of a conceptual architecture description to provide additional detail not readily available via the means of the diagramming solution leveraged for Figure 6.3.

6.3.1 Defining Platform and Analytical Tool

Some definitions are necessary before further description of the proposed SEM Selection Methodology. First of all, for a Platform for Systems Engineering Modeling and Planning, what is the platform? The answer to this question is stated as Definition 3:

Definition 3 The platform represents the set of analytical tools available in the MBSE ecosystem for conducting SE task planning analysis in the context of SE methodology formulation for writing a Systems Engineering Management Plan.

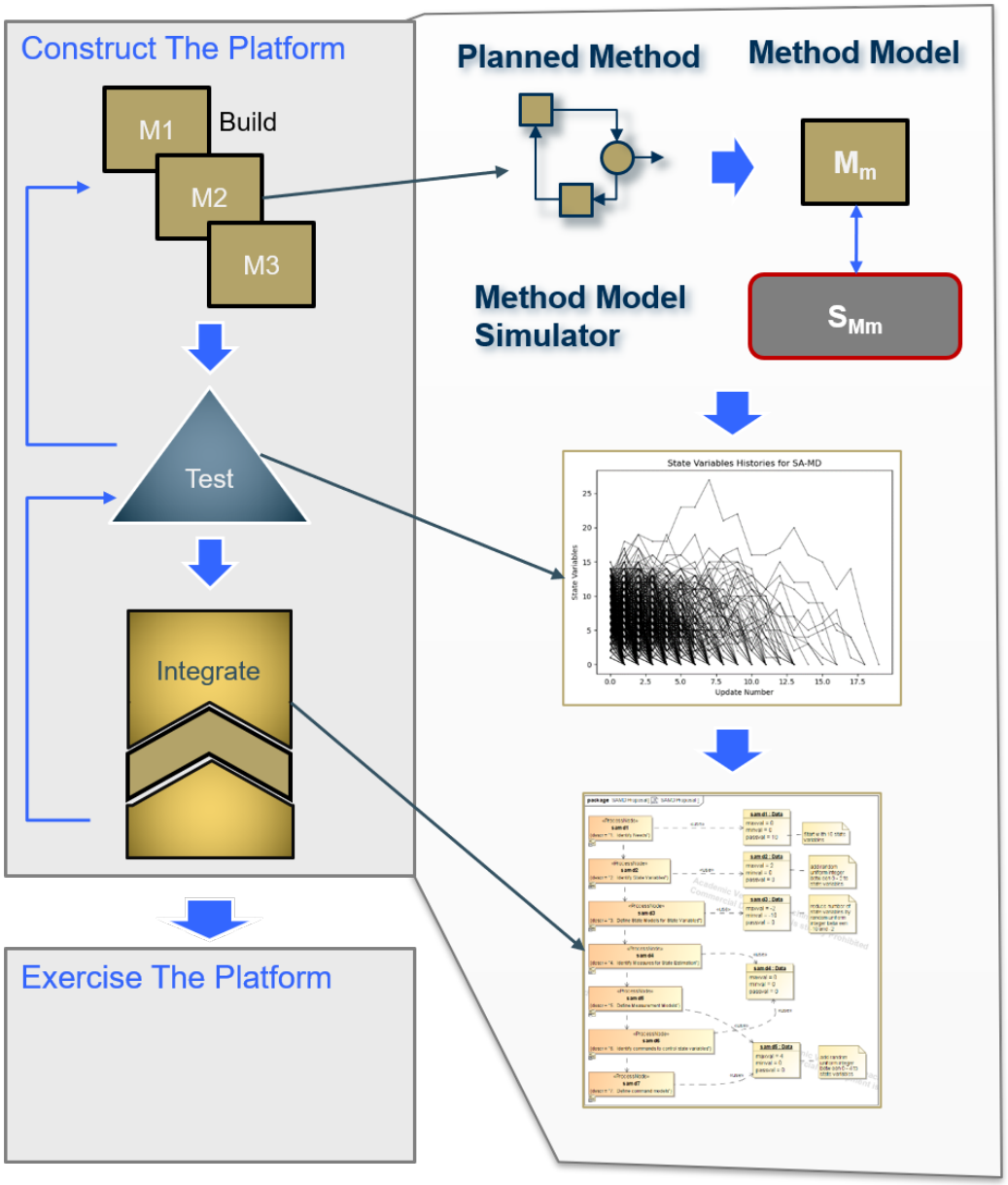


Figure 6.3: P-SEMP Methodology Overview

As such, the platform is not limited to just one particular incarnation. The primary innovation of P-SEMP as a methodology is the creation of this suite of analytical tools such that task proposals can be examined systematically. As described in Chapter 5, the idea of building P-SEMP consists of establishing various models and integrating them together. The latter portion of this dissertation will seek to accomplish this as described in Chapter 5, but as this is a *model-building exercise*, it is possible for

"exercise" of all possible aspects of the platform to necessitate future work (as will be seen in Experiment 4 the issue is the complexity of the problem). Effectively, through the philosophy described above, construction of the platform entails at some point the assertion of hypotheses on the existence, validity, or other claim regarding a model and testing thereof through simulation and calibration of the model, such that the hypothesis is accepted if the prototype succeeds in development or is rejected should the prototype fail. At the end of the experiments of this thesis, one such platform will have been established to some extent. This platform will enable simulation of Systems Engineering task models for comparison of method proposals through the exercise of the models which have been integrated to the MBSE ecosystem. However, conceivably some other set of analytical tools could be likewise arranged for such a purpose; in either case, the state of the art will have advanced beyond the assumed default described above, and the overarching methodology of P-SEMP will have been applied, whether the modeling focus parallels this thesis on the interest in task performance or whether some other aspect of PMTE (e.g. Knowledge, Skills, Abilities) is targeted for measurement, modeling, and simulation.

Additionally, it is important to consider what may be an analytical tool. Zeigler et al [56] was discussed in Chapter 3. In this foundational textbook for modeling and simulation, Zeigler et al provide a definition of the relationship between a source system, a model, and a simulator. Figure 6.4 is from [56] and illustrates the basic elements of the modeling and simulation environment. A portion of this figure has been removed for clarity as it refers to levels which Zeigler et al defined in a previous chapter. However, what is clear from this figure is the entities and relationships involved in modeling and simulation. Individuals engaged in modeling and simulation are interested in phenomena which are the data produced by a source system in some experimental frame (e.g. under experimental control) and which they seek to represent by abstraction via models, which are instructions which may be used by for

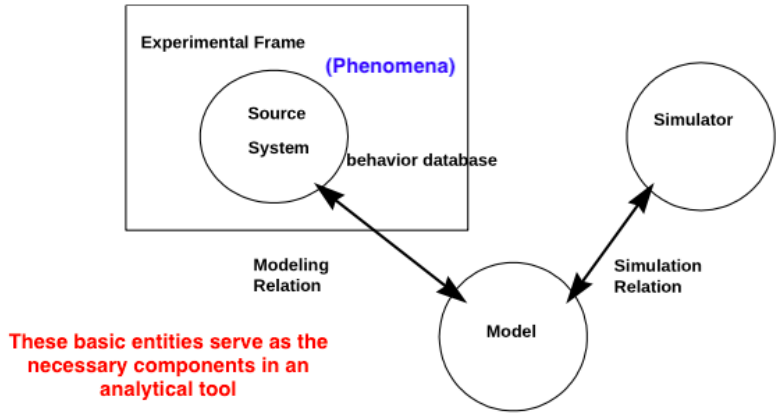


FIGURE 2.1

The Basic Entities in M&S and Their Relationships.

Table 2.1 Defining the Basic Entities in M&S and their usual Levels of Specification

Basic Entity	Definition	
source system	real or artificial source of data	Removed for Clarity
behavior database	collection of gathered data	
experimental frame	specifies the conditions under which system is observed or experimented with	
model	instructions for generating data	
simulator	computational device for generating behavior of the model	

Figure 6.4: From Zeigler et al[56, p. 28], Figure 2.1 and Table 2.1 illustrate the basic elements of modeling and simulation and how they relate. Some phenomenon and its data are modeled, thereby providing instructions which when simulated produces data which ought to correspond with the original phenomenon

example a simulator to produce new data which ought to have some correspondence to the original phenomena. This framework for understanding modeling and simulation is important, especially when models are lacking, and a need exists for new kinds of models as described above in the discussion of Morkevicius et al, Gilbertson, and Honour. For the purposes of this thesis, the investigation is more targeted, as defined by Definition 4:

Definition 4 SE Analytical Tools study SE Phenomena using a model and an appropriate simulator.

For planning purposes the phenomena will usually be related to process measures

or leading indicators, or likewise regarding task performance as defined by the tasks found in proposed SE methods.

6.3.2 Defining the P-SEMP Concept Architecture

In order to provide more granularity in the definition of P-SEMP as a methodology, a P-SEMP concept architecture has been created to help guide the understanding of P-SEMP and to track the proposed vs. as-built P-SEMP platform(s). Figure 6.5 provides an overview of all views related to the P-SEMP concept architecture. The first collection of views addresses the root concepts of P-SEMP. Several of these

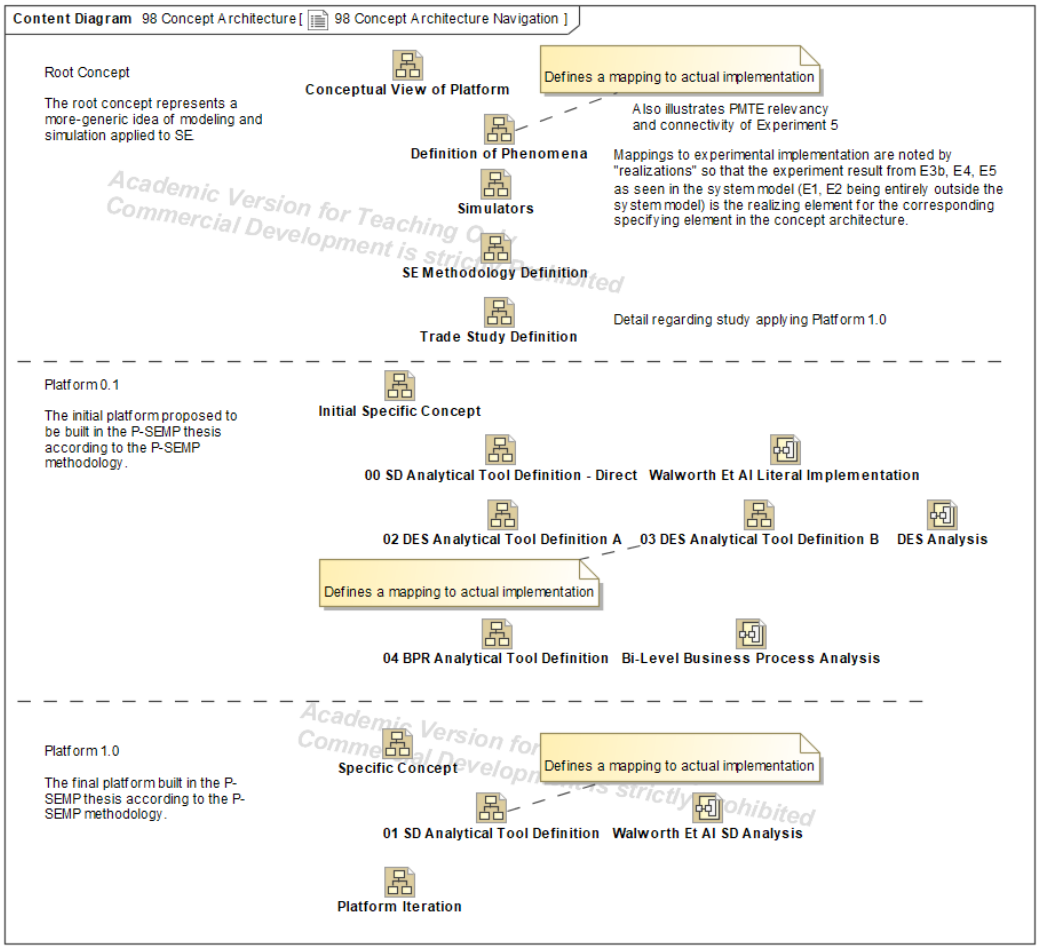


Figure 6.5: Overview of the Concept Architecture Views for the P-SEMP Concept Architecture

will be addressed shortly; however, issues like trade study definition will be revisited

later in the thesis after the experimental implementation. Subsequently, the intent of the proposed platform to be constructed by the experiments is illustrated by the views surrounding Platform 0.1. Platform 0.1 represents the "platform" of P-SEMP as proposed, at least initially, to address the specific gaps, questions, hypotheses, etc. currently under investigation. As mentioned in the definitions above, this does not rule out the possibility of additional different or similar platforms which accomplish synergistic objectives; perhaps by applying different analytical tools less interested in method performance assessment. Finally, the Platform 1.0 views illustrate the final state of P-SEMP and will be discussed further later in this thesis when the P-SEMP methodology is revisited.

6.3.3 Understanding the Pieces for Constructing the Platform

The first phase of P-SEMP in 6.3 is to construct the platform. That is, the Systems Engineering Management Team must construct the set of models suitable for representing their SEM proposal. To build these models, it is important to define the context in which they will be used and which phenomena they will address with which simulator. Additionally, the platform made up of tools using these models should be applied in SE trade studies by a SE planner to investigate SE methodology, and in particular SE methods. For that purpose a root concept architecture view is portrayed in Figure 6.6.

The development of simulations is a complex process. Due to the model relationships discussed by Zeigler et al [56] in TMS 2019, each of the model under simulation, model of the simulator, and representational model of the phenomena are important to consider, summarizing what is shown from that text in Figure 6.4 by Figure 6.7. That is, to develop the simulation models, the user of the P-SEMP methodology must first consider the phenomena which are to be simulated.

However, Figure 6.6 has several different kinds of relations between SysML blocks.

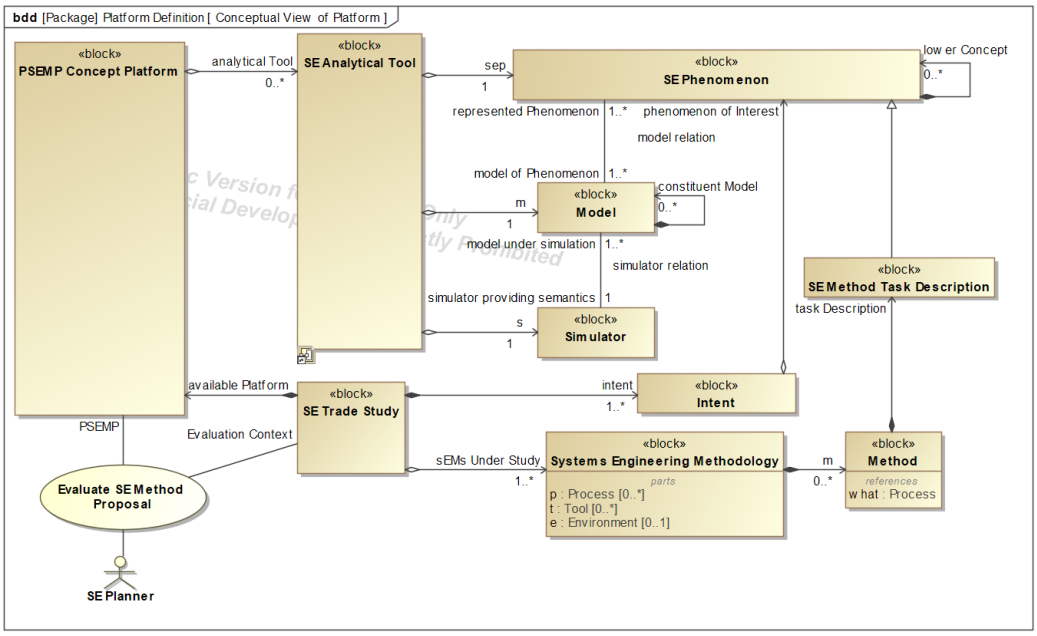


Figure 6.6: Root concept architecture view illustrating the relation of the analytical models with respect to a trade study on systems engineering methodology.

Some of the implications of these relations are addressed and a simple example is setup to help explain the root concept architectural decisions in the use of one relation versus another.

Explanation of the Relation Choices

The main sets of relations visible in Figure 6.6 include black diamond arrows (directed composition association), white diamond arrows (directed shared aggregation association), an arrow with a white triangular tip (generalization), and plain black lines (association). The generalization is the easiest to explain in simple terms: by reading in the direction of the arrow, the generalization arrow should be read as "SE Method Task Description" *is a* "SE Phenomenon". This statement of identity carries the implication that all properties and or behavior of the SE Phenomenon should be expected in the SE Method Task Description, or any other kind of SE Phenomenon, as a sort of taxonomic relationship. However, to explain other relations, a simple example of a brick house will be used.

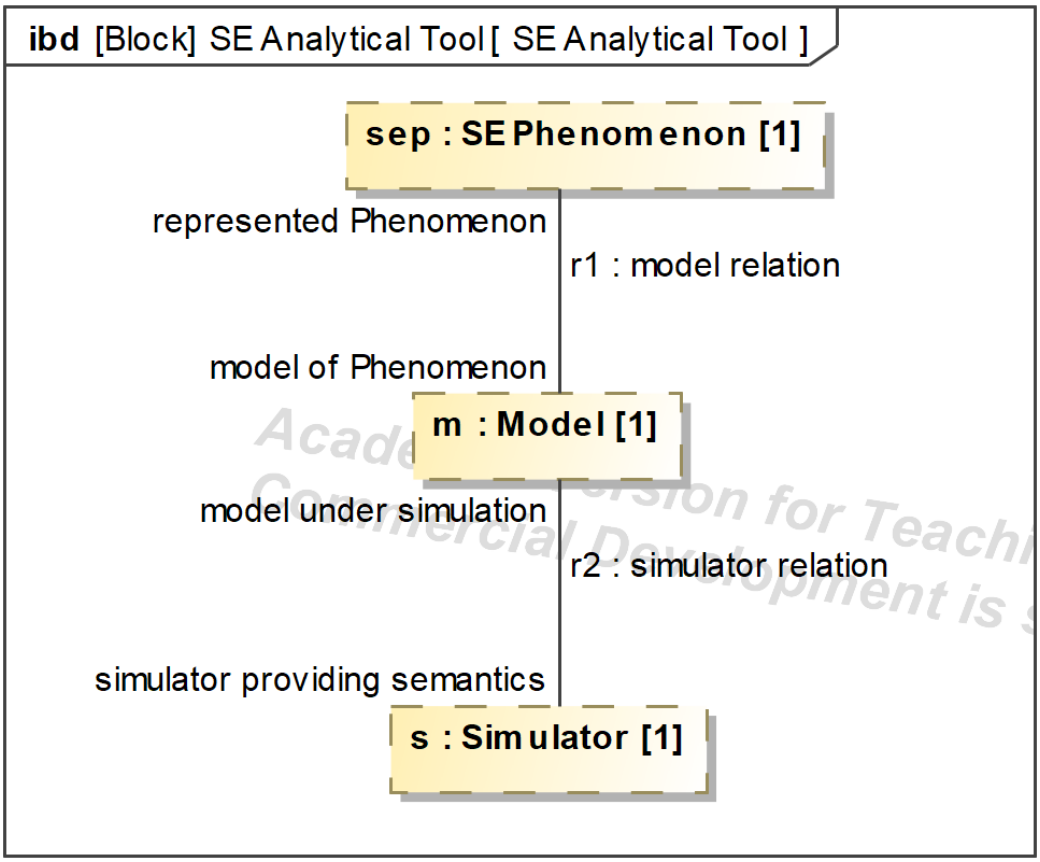


Figure 6.7: Model-System and Simulator-Model relations from Zeigler et al[56] mapped into the concepts relating to the SE Analytical Tool.

Brick House Example Illustrating Relation Implication

Figure 6.8 gives an example of a House and Brick, joined by a directed composition association such that House has a part "bricks", the value of which is defined as having 1 or more objects of type Brick. Note that this is different from a naïve reading of the arrow as in "House to Brick" or "House then Brick" which apply some sort of spatio-temporal relation on the arrow. In fact, there is no direct temporal nature

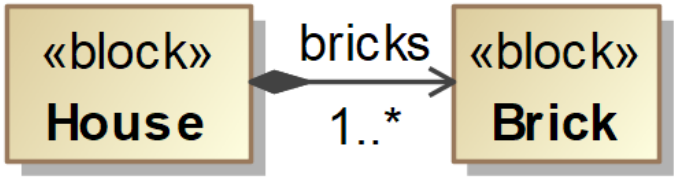


Figure 6.8: A house made of bricks

asserted between House and Brick, only that the objects making up the part bricks exist only in the context of House: more specifically, that the existence of this part bricks is fundamental to the identity of the House.

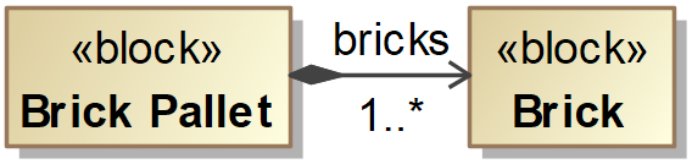


Figure 6.9: A pallet full of bricks

While it may appear at first glance that the only usage of Brick is in the context of House, it is possible to simultaneously define a Brick Pallet which also has a part bricks with 1 or more values of type Brick as in Figure 6.9. So now, in this conception, both Brick Pallet and House exist simultaneously. It is not clear which part bricks comes "first" as no notion of time yet corresponds between House and Brick Pallet other than in intuition. Consider perhaps, the analysis tool and the SEM definition of the conceptual architecture existing simultaneously and made up, at some point, of method descriptions as either the phenomena or likewise a method. In that case, there may arise the question of whether the method description only exists in the analysis context or just exists there first. However, in the house example, there is yet no spatial nor temporal ordering applied across Brick, House, and Brick Pallet. Instead, what would be needed under this simple example is a different kind of view, specifically Figure 6.10.

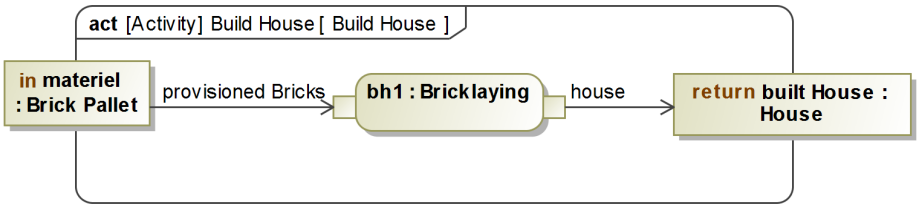


Figure 6.10: Build the house by laying the bricks from the brick pallet.

Figure 6.10 is a different kind of view than Figure 6.8 and Figure 6.9; it is an activity, which defines a process flow. In this particular process flow a brick pallet

enters as the materiel for building a house, and is the provisioned bricks for bricklaying. The action of bricklaying results in a built house, of course by removing bricks from Brick Pallet and placing them in the bricks of the House. This additional layer of description thus provides a temporal (but not spatial) relation between the bricks of the pallet versus the house. However, note that additional views outside of the simpler SysML blocks have been necessary to get to this point of description.

Concluding the Relation Description

Thus, enter the white diamond arrow (directed shared aggregation association) which indicates a usage of the item on the arrow end, without the necessary consequence of the "part" in terms of object value definition. That is to say, in order to avoid potential concerns on the uniqueness or primacy of the definition of SE methods as phenomena for the analytical tools in Figure 6.6, the SE Analytical Tool uses a shared aggregate of the SE Phenomenon, Model, and Simulator, so that their object values can be properly defined elsewhere and referred to (as a reference property) in the SE Analytical Tool. However, it is important to note that strict equality of the object values is defined separately from these arrows, and specifically for either style part or reference, black or white diamond, the relation called a binding connector would be required to actually assert equality between the object values, as shown for the simple brick house example in Figure 6.11.

For the P-SEMP conceptual architecture, the level of detail is not high enough to warrant specifying all potential equalities and all process flows. Instead, the guarantee that the SE Analysis Tool is not independently defining SE methods is the main concern.

Meanwhile, the plain lines (association) between the oval use case merely indicate some hypothetical involvement of the P-SEMP Concept Platform and the SE Trade Study in the conduct of SE Planner's evaluation of SE Method Proposals. Addition-

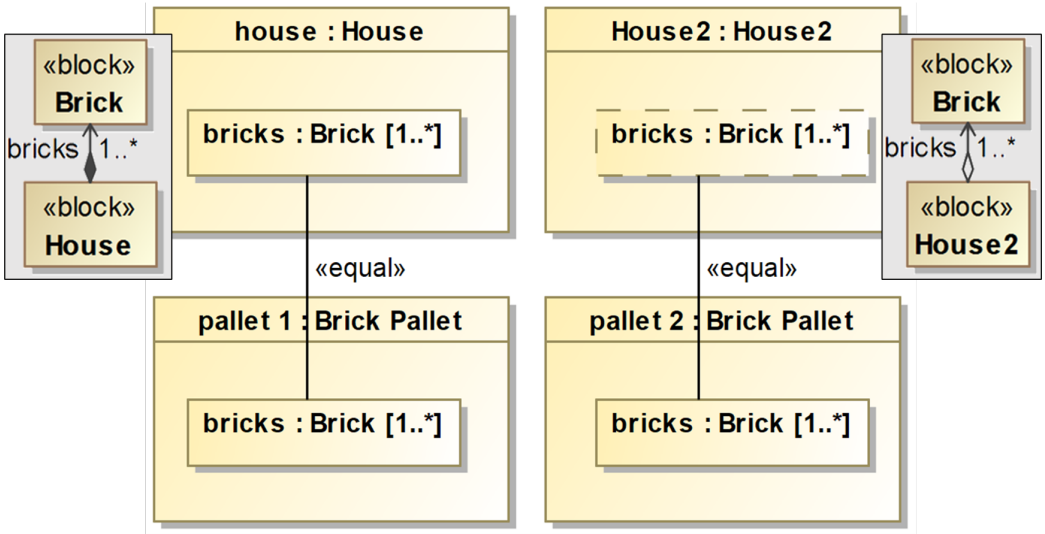


Figure 6.11: Asserting equality between parts and references within a context using binding connectors.

ally, these plain associations can indicate the roles between model, phenomenon, and simulator, and can be utilized internally to the definition of the SE Analytical Tools to help indicate to viewers what the relationship is between these pieces of the SE Analytical Tool. The relations described above will continue to be used throughout the concept architecture views.

6.3.4 Various Phenomena in the Concept Architecture

The concept architecture carries with it the definition of several ancillary pieces of information outside of Figure 6.6. Specifically, Figure 6.12 presents a wide array of content related to the subjects of various SE Analytical Tools. Clearly Figure 6.12 has many different elements and relations described. Of note in the purple region of the diagram are the SE phenomena which are studied by various future SE Analysis Tools which include Learning, Spacecraft Requirements Derivation Task Definition, Model Development Task Definition, and Robust Design Simulation Task Definition; the latter three being the tasks defined for three method proposals as part of the set of methods for three different Systems Engineering Methodologies. One SE method-

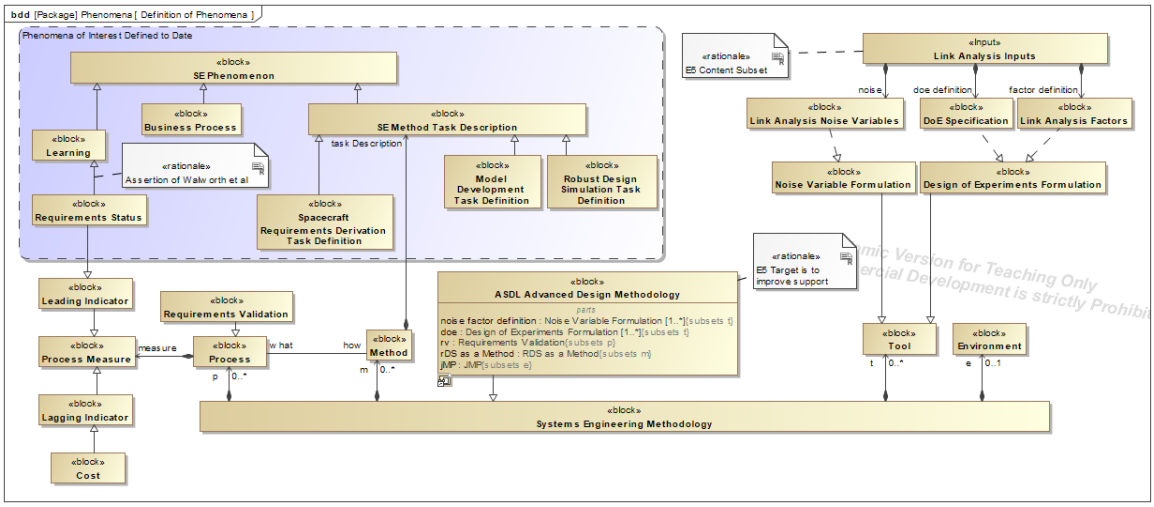


Figure 6.12: Phenomena which are studied by the current platforms and associated Systems Engineering Methodology structure.

ology in particular, for Robust Design Simulation, is given in Figure 6.12. The SE methodology representation uses the definition of SE methodology from Martin [4] and from Estefan [1] which defines Systems Engineering Methodology (internally) in terms of Process, Methods, Tools, and Environment. For the example, the ASDL Advanced Design Methodology is a Systems Engineering Method with two tools specified including noise factor definition and Design of Experiments Formulation, alongside the Robust Design Simulation tasks as a method which help to perform the process of Requirements Validation in the environment of JMP per the discussion in Reilley et al 2019[62]. Meanwhile, the right side of Figure 6.12 peels back the curtain on some of the items which are appearing just slightly ahead of schedule in the text of this dissertation; note that later in Experiment 4 and Experiment 5 this content will be discussed further. Suffice to say that the dashed line with white arrow tip is the

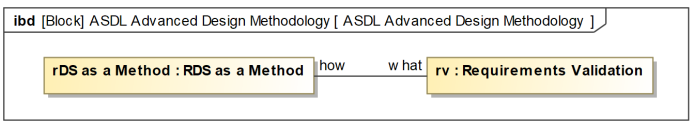


Figure 6.13: Martin[4] and Estefan[1] describe the process as what the systems engineer does, and the method as how the systems engineer does it. This structure is captured in the conceptual architecture description for P-SEMP.

realization relationship, indicating that the source element is realizing the targeted specifying element. In particular, what is shown here is that the results from the (at this point in the document) future Experiment 5 realize the specified tool description elements on Figure 6.12 for noise and DoE content. Finally, diving into the definition of the Systems Engineering Methodology block itself, Figure 6.13 illustrates the connection between a method and a process in terms of the how and the what, respectively, whereby according to Martin the SE Process is the standardized task of the systems engineer, whilst the method is how the systems engineer will accomplish the SE Process — likewise, the tasks of the method can be held in turn as a process, and an iterative or recursive definition of the workflow can be established[4]. Some of this discussion occurred earlier in the first chapter of this thesis, but is captured here in the conceptual architecture anyways.

6.3.5 Definition the Simulators of the Conceptual Architecture

Another aspect of Zeigler et al's [56] definition of the elements of modeling and simulation was the simulator. Figure 6.14 illustrates a set of simulators relevant to the current P-SEMP effort; however, there could well be additional simulators appropriate for P-SEMP. Specifically, this set of simulators does not exclude simulators which are not shown explicitly.

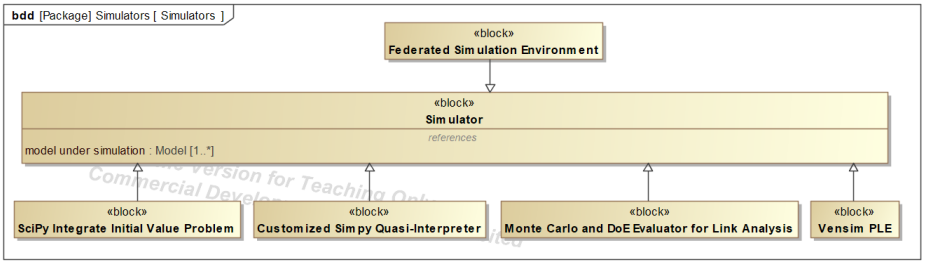


Figure 6.14: Simulators which are described in the P-SEMP conceptual architecture description.

In Figure 6.14, two simulators are directly related to the proposed SD modeling for Experiment 1. The SD modeling would potentially be a purely visual exercise, in

a simulator environment capable of turning the visual portrayal into a set of ordinary differential equations and performing the integration. However, it is also possible to write the equations directly if their form is known. Additionally, there is a single proposed environment for DES, although this is somewhat a factor of the results of Experiment 2 which will be seen later. Additionally, for Experiment 3a the potential for some sort of *federated* simulator which communicates between the SD modeling tool and the DES modeling tool might be required (e.g. as some sort of commercial solution), however as noted earlier in Chapter 3 Zeigler et al [56] would not require a *federated* simulator as necessary for combined DEVS-DESS which is effectively the proposal for Experiment 3a.

6.3.6 SE Methodologies Under Consideration

Finally, as far as root concepts go for the P-SEMP conceptual architecture, there is the set of expected Systems Engineering Methodologies which are studied in this thesis. Figure 6.15 illustrates the setup of some fragments of these methodologies.

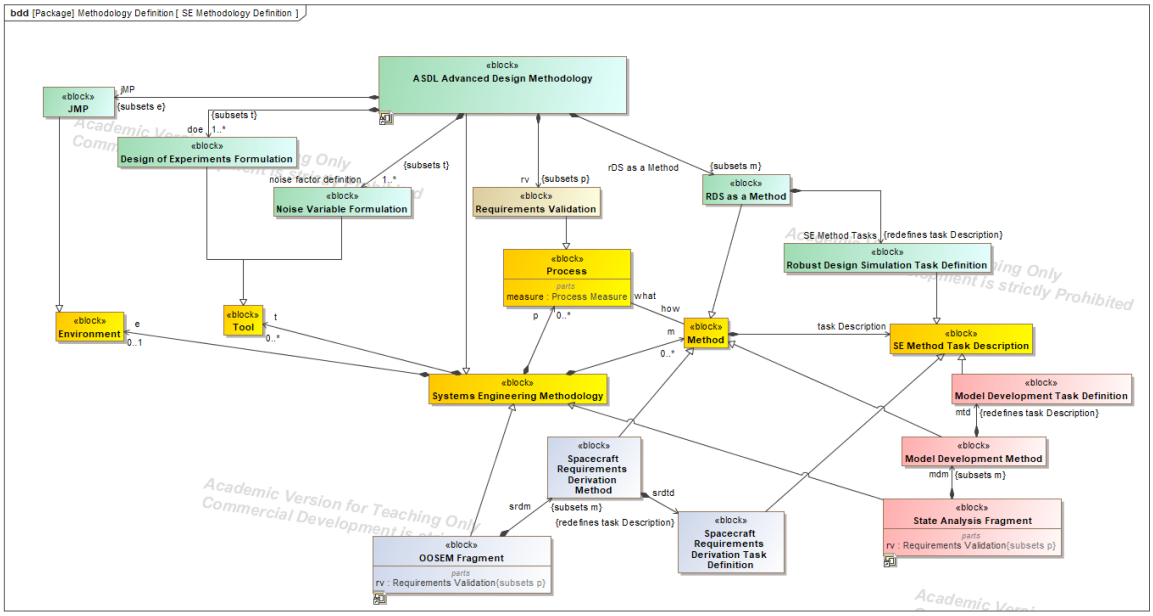


Figure 6.15: Methods described within the P-SEMP thesis and subject to the P-SEMP methodology.

Figure 6.15 shows three proposed Systems Engineering Methodologies including the two described in Chapter 5, OOSEM-Lite from Friedenthal and Oster[98] and specifically the Spacecraft Requirements Derivation step, as well as State Analysis Model Development [21]. None of these three methodologies are described exhaustively by the content of Figure 6.15, merely specific features of the methodologies which are relevant in the P-SEMP conceptual architecture and for the conduct of the experiments of this thesis.

6.4 Constructing the Proposed P-SEMP Platform Version 0.1

The proposed experiments in Chapter 5 focus on the development of a particular kind of integrated modeling environment for these proposals, a kind of systems engineering shop floor inspired by the business process literature as described by Jahangirian et al[46]. In the proposed experiments, leading indicators are emphasized due to the alleged predictive capability that they provide, and models of leading indicator evolution are considered critical for planning purposes. Thus, a requirements status leading indicator model from Walworth et al is proposed as a baseline model, within which or alongside of which specific process-oriented models representing the task proposals are to be constructed and simulated by e.g. DES. If successful, these task proposal or method models for the method proposals will be incorporated to the Walworth et al model, providing a hybrid and bi-level business process simulation environment for SE methodology.

Figure 6.16 illustrates the proposed platform to be constructed by the P-SEMP thesis in the proposed experiments. As part of the conceptual architecture, Figure 6.16 specifically introduces the SE Analytical Tools to be incorporated into this particular incarnation of a platform. Each model to be built may have an associated SE methodology phenomenon, modeling formalism/representation, and simulation environment or model of simulator. For example, the phenomena illustrated on Fig-

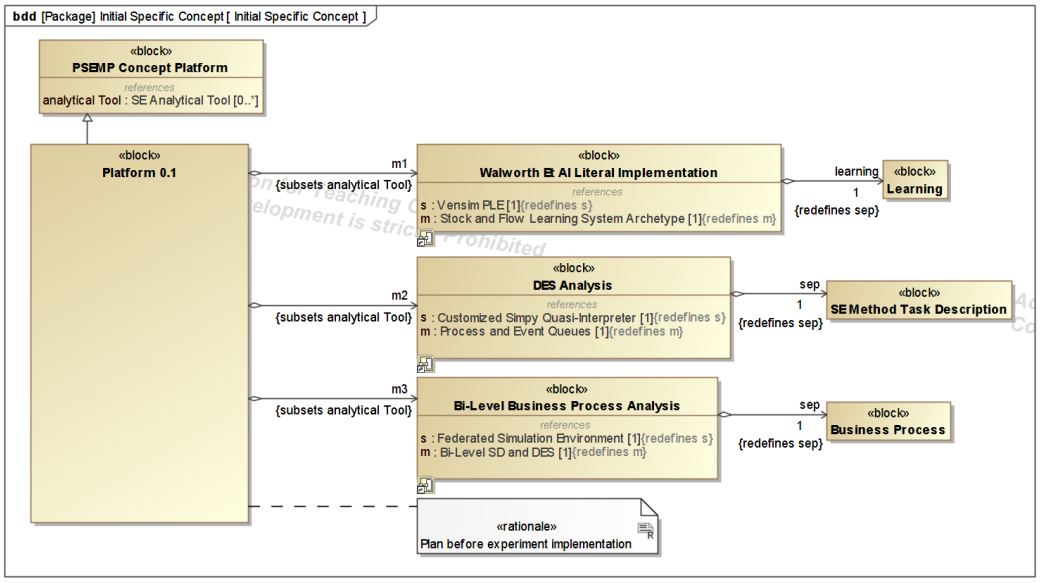


Figure 6.16: Proposed Platform Conceptual View: Version 0.1

ure 6.16 including Learning (as a proxy for requirements status), SE Method Task Definition, and the Business Process (of the SE enterprise).

Note that this particular proposed configuration is just the most promising set identified for this incarnation of the platform. Other models may be identified as of primary importance by other groups based on their measurement, modeling, and performance objectives; the key is to have the simulation models available for conducting proposal assessment on an improved quantitative basis over the ad-hoc approach. In the experiments of this thesis, DES stands out as a technique for capturing the task proposals and simulating task architectures in the method models for analyzing which plans might be better suited to the objectives of the organization. This will be elaborated on over the course of Experiments 1 - 3, and in part Experiment 4. The Systems Engineering Management Team may apply a domain-specific language to represent the simulation models and plan in the system model. The team will conduct calibration studies, and determine the range of validity of their models given the data their organization has collected over previous efforts. As the platform construction is the primary emphasis of this thesis, further description is provided below.

6.4.1 Development of Simulation Models

In the case of this incarnation of P-SEMP, the phenomena are the tasks proposed in methods for SE processes as part of SE methodology planning, assessment, or selection/comparison. For example, while such a quantitative comparison provides objective improvement over ad-hoc methodology comparison, the phenomena related to tasks also play a role in planning and mitigation actions during the conduct of systems engineering work — thus the platform serves a recurring purpose during the system life cycle in assisting the planning of SE activities if it is constructed of models representing the appropriate phenomena. Subsequently, there are the models under simulation, the mathematical or programmatic depiction of the particular phenomenon as appropriate. Finally, a simulator is a model of how to compute the results or consequences of the model under simulation. Sometimes, the simulator model is provided by a commercial tool, for example if using the Mathematica language, the simulator model is baked into the parsing or compiling of the language while the language itself is constructing the model under simulation according to specific understandings or representational models of phenomena. Likewise, in DES, a open tool such as simpy may provide simulation semantics for DES in python and thus enable the construction of a simulator environment. However, using this kind of open tool requires a bit of extra configuration to specify the precise simulation model and potentially contigent models under simulation. Experiment 2 documents in detail the establishment of this kind of simulator-simulated model environment. Experiment 1 documents a set of related efforts surrounding the Walworth et al model. Due to the visual nature of SD modeling, a tool which provides the visual capabilities and simulator model (Vensim PLE) is first used; after which, python-based ODE solving is applied to the simplified differential equation formulation developed in Experiment 1. These model-building efforts are necessary to be able to select models for integration to the systems engineering ecosystem for use in planning activities, and if

successful Experiment 3a would conclude the model building. However, Experiment 3b continues with the integration to the systems engineering ecosystem. The models developed are to be integrated to the ecosystem for use in planning. The proposed analytical tool integration in the systems engineering ecosystem provides a platform for systems engineering modeling and planning, as seen in Figure 6.16.

Proposed SD Model Analytical Tool

The original proposal for the Walworth et al model in the P-SEMP thesis according to the conceptual architecture description is presented in Figure 6.17. This SE Analytical Tool was proposed to be established by Experiment 1 outside of the MBSE environment and later incorporated to the MBSE environment in Experiment 3b, establishing it as part of the implemented Platform.

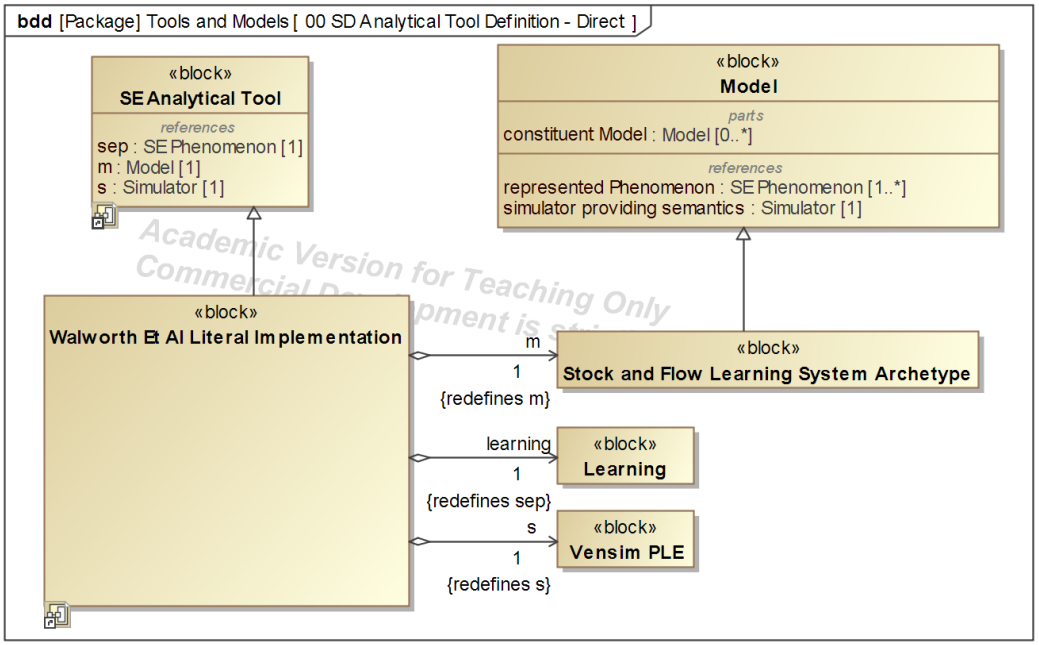


Figure 6.17: Proposed Experiment 1 Result

Internally, the SE Analytical Tool as proposed for Walworth et al establishes the modeling and simulating relations as expected, and as shown in Figure 6.18. This provides a level of completion of the conceptual architecture for documenting how

these various pieces of the SE phenomenon of Learning, together with Vensim PLE and the Stock and Flow model come together for the proposed result.

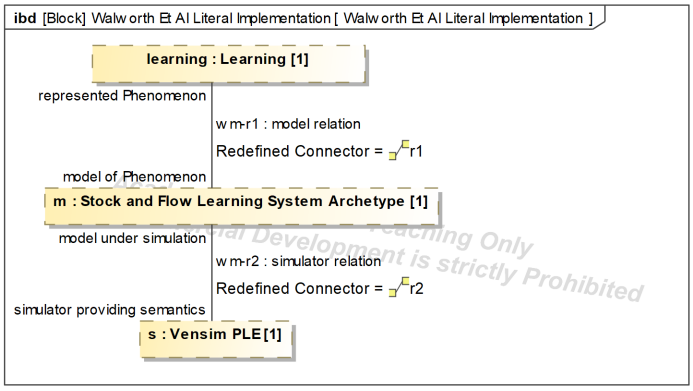


Figure 6.18: SD Modeling and Simulation Relations

Of course, the actual implemented platform and analytical tool may differ. These differences will be addressed in the course of this thesis.

Proposed DES Model Analytical Tool

The second SE Analytical Tool proposed Platform version 0.1 is the DES capability focused on the SE Method Task Descriptions. Figure 6.19 illustrates the proposed SE Analytical Tool. The analytical tool itself is built over the course of Experiments 2 and 3b, based on the results of Experiment 2 in terms of the Simulator and Model.

As before, the internal relations of the phenomenon, model, and simulator are established for the conceptual architecture description. This internal relationship is illustrated by Figure 6.20.

Note that the intent of Experiment 2 is to configure the environment described for DES such that the SE Method Task Descriptions correspond flexibly to different method proposals. Thus, in the conceptual architecture, the phenomenon is left at the more generic level rather than the more specific level necessary for the SE methodology fragments of Figure 6.15.

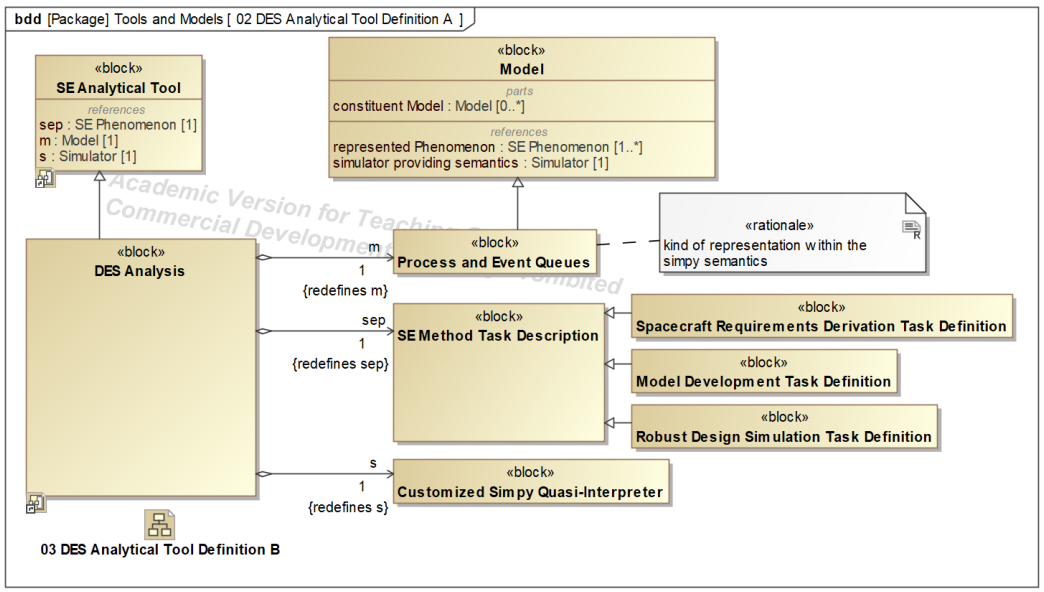


Figure 6.19: Proposed Experiment 2 Result

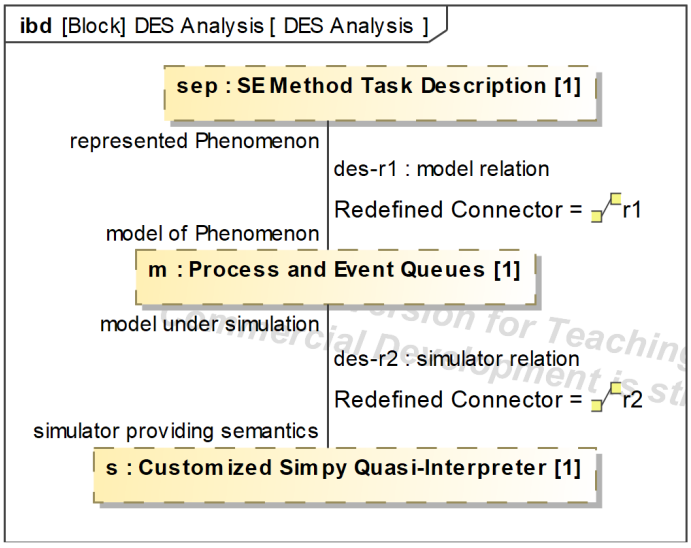


Figure 6.20: DES Modeling and Simulation Relations

Proposed Business Process Model for Hybrid Simulation

The final analytical tool in proposed Platform 0.1 is the hybrid simulation capability inspired from the business process domain as described by Jahangirian et al [46]. This capability was proposed for Experiment 3a. Figure 6.21 illustrates its place in the conceptual architecture.

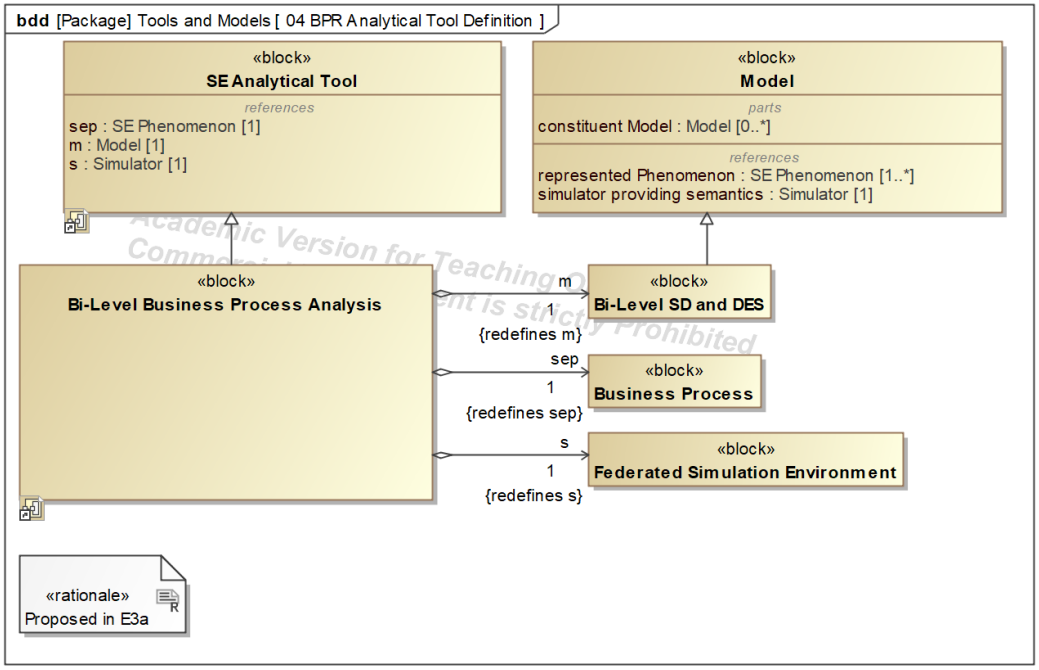


Figure 6.21: Proposed Experiment 3a Result

As before, the internal modeling and simulation relations of the proposed hybrid business process simulation for Experiment 3a are illustrated by Figure 6.22.

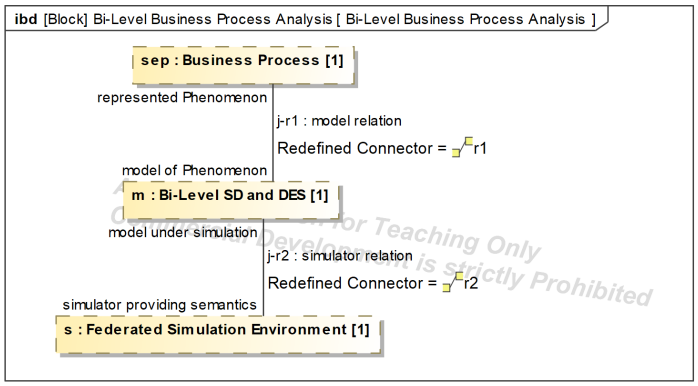


Figure 6.22: Business Process Modeling and Simulation Relations

What to do with the constructed platform?

In the end, this platform is intended to be exercised to analyze task proposals and enable decision-making based on the quantitative outputs of the simulation models, whether for SE methodology selection or for mitigation action design during the

course of SE activities.

6.5 Exercising the Platform

The second phase of P-SEMP is to exercise the platform. To exercise the platform, the Systems Engineering Management Team is interested in first planning SE activities, as well as monitoring SE performance. To this end, the models which have been constructed as part of the platform are simulated to determine appropriate bounds for SE performance parameters for organizational and program objectives. Experiment 4 touches on this aspect of exercising the platform. When selection has been completed, the management team is able to set targets on performance parameters so that they can be measured and monitored as the team does its work. However, if some tools or aspects of the environment are as yet unavailable, then task performance may not be able to meet that which was modeled. For this purpose, Experiment 5 develops some capabilities necessary towards one of the methods investigated. Overall, if performance is falling behind, the team can use its models and perhaps explore the task architecture space to determine the best remedy to accelerate the SE performance without losing quality in the process. One example of exploration will be given later in revisiting the conceptual architecture shown here, based on the result of Experiment 4. However, any exercise of the models will use some form of the SE Trade Study from the bottom of Figure 6.6.

6.6 Summary

The P-SEMP methodology is focused on bringing measurement and rationality into the consideration of SEM. The objective is to determine better ways of doing SE. For this purpose, task proposals are to be modeled in such a way that process performance can be forecasted. These forecasts along with subjective criteria can then

aid and improve decision-making for SEM selection and usage, as well as provide a mechanism to intervene in the course of product development. This chapter has elaborated on the philosophy underlying the P-SEMP thesis, as well as the P-SEMP methodology especially in terms of a conceptual architecture description to elaborate on the elements and relationships crucial to the experiments and later, their results.

7

EXPERIMENT 1: REPLICATION OF THE WALWORTH ET AL. SYSTEM DYNAMICS MODEL

The purpose of Experiment 1 is to provide a canonical model for requirements volatility that can be tightly integrated with a system model representation. This canonical model can provide a basis for comparison against new methods of simulation for SE performance. Specifically, the intent is to investigate identification of important parameters involved in requirements status and how this status evolves.

7.1 Recap of the Experiment

The task of replicating Walworth et al[42] primarily serves to help provide a basis against which it might be possible to better understand parameterization of requirements status, according to Hypothesis 1a:

> If requirements status is parameterized, then important parameters can be identified and understood.

The experimental procedure described earlier in this thesis for Experiment 1 involved the following tasks:

- Construct System Dynamics (SD) Model in modeling tool
- Vary model parameters to obtain s-curves from literature
- Identify possible parameters to be calculated by method models
- Explore integration with a system model

However, the conduct of any experiment is rarely straightforward, especially in model-building. Of note, one of the concerns raised by Gap 3 is due to the nature of the parameters given by Walworth et al for their requirements status metric. While a SD model is quite capable of generating the s-curves representative of requirements measure tracking, their parameters included "soft"-er terms such as "Effort", "Intensity", and "Learning Power", terms which are not precisely defined[42]. Thus the key concern in this replication study is to be able to run the Walworth model for the purpose of investigating how these kinds of parameters might be better-defined in terms of SE methodology.

As Experiment 1 is essentially a model-building exercise, it will involve a set of iterative steps to construct the model. In practice, this involves a series of assumptions as working hypotheses which are accepted or rejected as the model is validated. The end result is expected as a working prototype; however, each assumption in how to build the model represents in part a form of hypothesis, which is affirmed or rejected according to the success of the prototype development.

7.2 System Dynamics Replication

This replication study will attempt to rebuild the SD model presented by Walworth et al[42]. At each step, the relations from the journal article will be assumed as correct and the implementation will be attempted in a visual SD modeling tool: Vensim PLE. Each assumption is effectively a primitive form of hypothesis against the prototype. If each step succeeds without issue, then all of these prototypical hypotheses are accepted and the prototype will match the journal article model. However, this chapter will attempt to avoid the pedantic nature of writing out each of these assumptions (proposed relations) as formally identified hypotheses. The following describes this procedure and reported results regarding the replication study.

7.2.1 Defining System Dynamics Models

SD modeling proponents argue that their approach to representing and simulating complex systems features more qualities of *Systems Thinking* than alternative modeling approaches. According to McDermott[105], *Systems Thinking* consists of "Sensemaking... Goal setting... Model selection... Feedback". McDermott realizes the system model by leveraging causal loops to represent the feedback system, seamlessly showing SD as integral to *Systems Thinking*[105]. More generally, Mobus and Kalton describe *Systems Thinking* as "the approach of applying principles of systems science" [p. 81], while describing systems science as a "metascience," which "is a way to look at all parts of the world in a way that is unifying and explanatory"[p. 5][106]. Morecroft is more confrontational, arguing that the antithesis to *Systems Thinking* is event-driven thinking[107]. Morecroft insists that the natural approach to modeling complex systems across many levels of detail is the SD approach. In this case, Morecroft describes SD models as representing the behavior of complex systems via the structure of their stocks, flows, and especially their feedbacks which are the necessary component for driving the often counter-intuitive results of famous SD models in the historical literature[107]. From these works, a few key details are apparent:

- SD model is a set of integral-differential equations represented visually
- SD model utility is not purely in the equations
- SD model utility lies moreso in the visual representation and model-building activity with stakeholder involvement

These concepts indicate two possible understandings of the Walworth et al model. The first approach would be to utilize a SD modeling tool to recreate the Walworth model, for the purpose of the visualization of the model structure and the interactive model-building purposes of SD practice. However, for rapid exercise of the model, an

approach leveraging the equations without the visual features may be more suitable depending on particular software features available.

In the first approach, a decision must be taken regarding the visual SD tool. There are several tools from which a selection might be made. For the purpose of this experiment, the visual tool selected was Vensim PLE, and the tool for running many cases on the differential equations was Python code using PyDOE2 and Scipy.

7.2.2 Initial Successful Procedures

When re-creating the model from Walworth, there are some considerations which must be accounted-for in the drawing of the model structure. For example, a tool may not accept the chaining of two flows together, as if aggregating derivatives. This process is prominently displayed for the learning system element of the Walworth model, where the diffusion model has multiple chained flows from the Tasks to be Done, to the Tasks Really Done and Task Done Wrong stocks.

In investigating the creation of the model, each element can be considered a form of hypothesis, that in fact the element is representative of something in the world. Here, a sub-hypothesis will be that an intermediate stock for managing the coupling of the flows is equivalent to what is depicted in the Walworth model, and is compatible with the selected visualization tool. For this added stock it is trivial to see that the result is as intended:

$$\text{Work In Progress} = \int \text{learning rate} - (\text{work done right rate} + \text{work done wrong rate})$$

A further assumption would be a working hypothesis that the Walworth "misunderstanding rate"[42] is representative of the intended aggregation of the learning rate with the work done wrong rate. That is, the two rates indicated sequentially

on the flow from the stock Tasks to be Done to Undiscovered Rework are multiplicatively combined, where the intended "work done wrong rate" is just the multiplicand $(1.0 - \text{Quality of Work Done})$[42].

Thus, with the introduction of the intermediate stock, a simple revision is necessary for the work done X rates. Providing an arrow from learning rate to the affected rates enables the use of learning rate directly in their equations, and thus the revised "work done wrong rate" can be set equal to the misunderstanding rate in Walworth. As a consequence, this means that the equation for the Work in Progress stock takes the form:

$$\begin{aligned}
\text{Work In Progress} &= \int \text{learning rate} - (\text{work done right rate} + \text{work done wrong rate}) \\
&= \int \text{learning rate} - \text{Quality of Work Done} * \text{learning rate} \\
&\quad - (1.0 - \text{Quality of Work Done})\,\text{learning rate} \\
&= 0
\end{aligned}$$

Thus, the intermediate stock Work in Progress is shown to have no affect which would alter the behavior of the model from Walworth, other than numerical artifacts which could arise from additional floating-point operations.

Furthermore, while not explicitly defined in the Walworth paper, as seen above work done right rate is defined by Quality of Work Done $*$ learning rate. The lack of explicit definition appears to be based on a tool-oriented comprehension of the stacked rates on a given flow, however the chosen relation is trivial enough to not warrant much discussion.

All other relations in the learning system/diffusion portion of the model are taken as-is, and assumed to be correctly formulated. However, more serious problems begin to arise with the re-work discovery.

7.2.3 Issues Stopping Full-Implementation

Walworth et al choose for error discovery "the Jelinski-Moranda equation... due to its success in the modeling of many data sets"[42]. An interesting observation with respect to the perspective of Morecroft on SD modeling is that the error discovery model used here is event-driven in nature. According to Jelinski and Moranda, define subsystems reliability[108]:

$$R_K = e^{-\lambda_E t}$$

Where R_K is subsystem reliability, λ_E is the subsystem error rate, and t is the subsystem mission time. Further, Jelinski and Moranda provide a model to account for the "intensity" of testing[108]:

$$\begin{aligned} R_K &= [N-(i-1)]\,\phi e^{-[N-(i-1)]\phi X_i^l} \\ X_i^l &= \tau_i - \tau_{i-1} \\ \tau_i &= F(t_i) = \int_0^{t_i} E(u)du \\ \phi &= \frac{n}{NT - \sum(i-1)X_i} \\ T &= \sum X_i \end{aligned}$$

Where N is the initial error content, ϕ is a proportionality constant (both N and ϕ are time-independent unknowns not affected by time averaging), X_i are the time interval samples between successive errors, Importantly, $E(u)$ is the exposure rate for the process, and relations for *phi* and N can be found by maximum likelihood with X_i substituted as X_i^l. The exposure rate is what gives the next time for an error

detection event.

The relation to be solved for N is, according to [108]:

$$\frac{1}{N} + \frac{1}{N-1} + ... + \frac{1}{N-(n-1)} = \frac{n}{N - \frac{1}{T}\sum_{i=1}^{n}(i-1)X_i}$$

The resulting estimate for N is called $\hat{N}$ and estimates ϕ as $\hat{\phi}$ by[108]:

$$\hat{\phi} = \frac{n}{\hat{N}T - \sum(i-1)X_i}$$

According to Walworth [42] this model is used to calculate a rework discovery rate and can be normalized by time, and then parameterized by urgency, intensity, attention span, and the number of potential rework tasks. However, the key issue in making use of the model is the functional form for $E(u)$, which is used in variable substitution for determining X_i. Walworth does not provide any further detail other than the citation of the original paper. However, the original paper by Jelinski and Moranda [108] asks users to establish their own exposure rate formula: "The approach is to employ $E(t)$ as a normalizer of time; more specificially[(sic)], $E(t) = 1$ for any period of normal exposure. A normal exposure, of course, is defined in terms of the specific problem at hand..."[108]. Walworth must have created such a definition of the exposure rate to use the model, and subsequently not reported the functional form used. In this way, it becomes impossible to reconstitute directly the equation relating Walworth's parameters for rework discovery according to the cited material.

7.2.4 Attempts to Extract the Discovery Relation

When it was found that the precise formulation of the model was not forthcoming, in that Jelinksi and Moranda provided no insight into how Intensity, Urgency, and

Attention Span might be formulated into the time transformation for X_i, the next attempt made was to investigate whether digitizing the plots in Walworth et al could provide insight on the relation. Plot digitization was performed using the GNU "Plot Digitizer" JAVA application on Ubuntu 18.04. Each individual time series on the plots for Urgency, Intensity, and Attention Span were digitized for each level of the variable displayed on the chart. The result is 15 datasets specifying a rework discovery rate as a function of time for each variable at a particular setting. However, what is missing in Walworth et al is any statement about the settings of the other variables for any particular setting of the variable of interest[42]. That is to say that Walworth et al report, for example, a timeseries line for Urgency = 10, but do not indicate the corresponding values for this trajectory of Attention Span and Intensity at any point in the journal paper[42]. Additionally, the plots do not specify that they are displaying anything like a partial derivative of the Rework Discovery Rate; they are specifically the Rework Discovery Rate response at various levels of the indicated parameters. Therefore, it is found to be impossible to reconstruct a combined function based only on the plot data which determines the Rework Discovery Rate from each of Urgency, Intensity, and Attention Span.

7.2.5 Discussion of the Canonical SD Model Representation

Despite the inability to run the simulation due to the missing Rework Discovery rate relation, the overall visual model is relatively straightforward to setup. The result of creating the model can be viewed in Figure 7.1.

In this figure, the result of splitting the flows for Work Done Right and Work Done Wrong can be seen, along with the "null" stock added to the model, "Work In Progress," as discussed in the earlier sections.

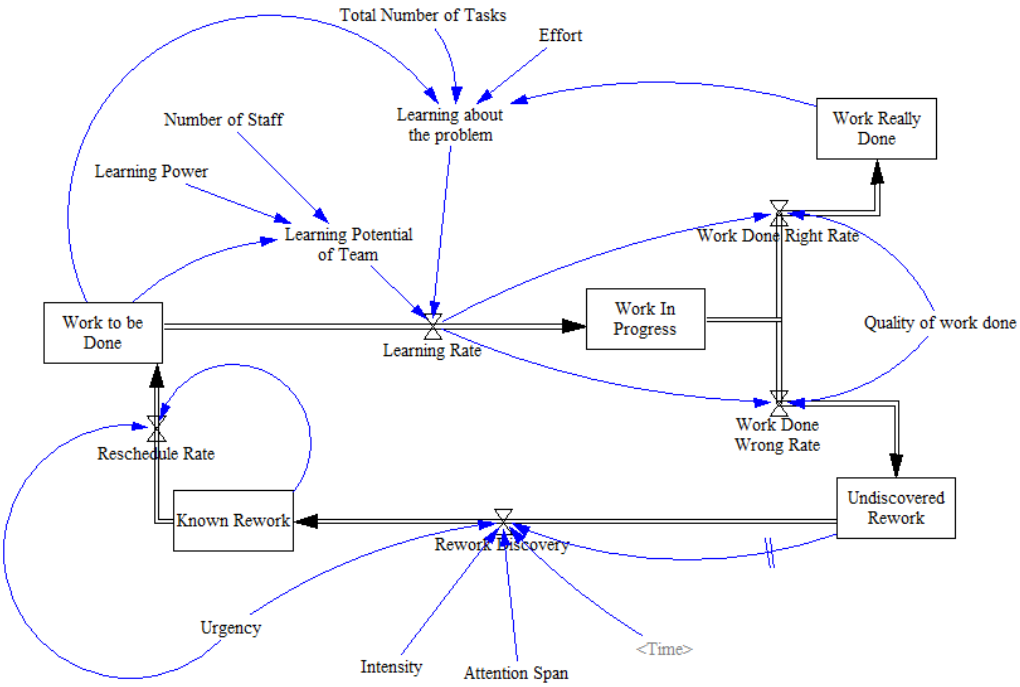


Figure 7.1: Walworth et al.[42] model reconstructed in Vensim PLE with an additional Work-In-Progress stock.

7.2.6 Formulating the SD Model in Python

Practically speaking, the visual representation of the SD model is sufficient for most soft-systems purposes. The visual representation of the causal links aims to assist stakeholder consideration of the system model. However, while some tools enable automated trade study evaluation, a key necessity in connecting the system dynamics model parameters is to have low-level access to the Application Programming Interface (API) for the program used in establishing the model. For this purpose, the differential equations defining the model were extracted and represented in Python for solution by numerical integration.

Of particular interest, one relation was modified between the models. The Rework

Table 7.1: Table of SD model variables default values, as well as minimums and maximum values used in the Design of Experiments (DoE) formulation later.

Variables	Default Value	Minimum	Maximum
Intensity	0	0	10
Learning Power	1	0	2
Number of Staff	2	1	10
Total Number of Tasks	100	50	200
Effort	0.5	0.01	2
Quality of Work Done	0.8	0	1
Urgency	5	0	10
Attention Span	0	0	10
Discovery Factor	5	0	100

Discovery Rate is now specified as:

$$Q = f * U$$

Where Q is the Rework Discovery Rate, f is an arbitrary rework discovery factor, and U is the Undiscovered Rework quantity in the stock. Importantly, in the implementation none of the levers are hard-coded. Default values are available, but otherwise the model can be run for any numerical values of the parameter set. Table 7.1 shows the default parameter definitions. Note that the initial stock values and time span are in fact hard-coded; however, during initialization of the model object, the value for the parameter total number of tasks is allowed to override the initial value for the work to be done.

SciPy Integrate for the Initial Value Problem is used to solve for the time-history behavior of the model. The entire model is wrapped to run according to cases specified by an input file determining appropriate settings for the parameters for each integration run. Attempts were made to parallelize cases; however, it appears that some significant modifications of the simple implementation would be necessary to enable proper parallel execution. Meanwhile, the single-threaded approach is still quite fast for hundreds of cases, running in a few seconds on a dated laptop with an approx-

imate i5-4260U 1.4 GHz CPU, 4 GB of RAM, and SSD. Modern hardware can run the cases even faster, although SSD speeds may provide limits due to file operations and compression using the npz format. The source code of model implementation is included in Appendix A, Section A.1.

7.3 Exploring Variability

In order to explore variability in the behavior of the SD model, it is necessary to sweep across different values of the input parameters. The four outputs of the model are the time-histories of the work to be done, work really done, undiscovered rework, and known rework.

7.3.1 DoE Definition

In order to systematize the exploration of the SD model, a Design of Experiments (DoE) is established. First, a set of ranges were defined for the default parameters, shown in Table 7.1. These ranges (min, max) are intended to set the bounds of the study to provide some understanding of the model behavior. A class was established for Design Factors, in order to track the minimum, maximum, and name attributes as well as manage normalization and de-normalization routines. Using the PyDOE2 library, a Box-Behnken design for 9 factors with one center-point was utilized, with the assumption that at most quadratic least-squares regression would be applied for examining the parameter sensitivity. The Box-Behnken design uses three levels: -1, 0, 1. The normalization scheme assures that for level 0, the value is in the middle between the minimum and the maximum of the variable range. The source code of model implementation is included in Appendix A, Section A.1.

7.3.2 Data Processing

The DoE data was saved as npz format in the de-normalized state. Due to using Python version 3.8+, and coordinating the definition of DEFAULTS with DEFAULTS_DOE, the columnar ordering of the variables is preserved so that the output of the DoE script is a table where the columns are in the order of the default parameters and the rows indicate the cases to be run. After running each case as an individual object of the Walworth model class, the list of objects (data and model behavior for each case) is saved to a Python pickle in npy format. These saved data can be loaded and a special-purpose plotting routine [109] can be used to visualize the data. In the default settings for the DoE, model, etc, the total amount of data is approximately 325 KB, excluding one 91 KB svg image. A portion of the run-time and a portion of the data is made up of profiling data, instrumenting the simulation to understand the costs associated in running the simulation. The source code of model implementation is included in Appendix A, Section A.1.

7.3.3 Results

Figure 7.2 shows the set of data for "Work to be Done" for all DoE cases. These figures provide an overview of the available dataset. Additional, an example of one particular case is given by Figure 7.3, for case 100. Additionally, figures of the form provided by [109] and seen in Figure 7.3 are given in Appendix B for all 145 cases of the DoE for the Walworth model.

7.3.4 Reduction and Fitting

Normally, surrogate modeling might be effective to provide simple models for exploring the system performance. In this cases, "system" refers to the SE enterprise. However, as these responses are time histories, the surrogate modeling activity may

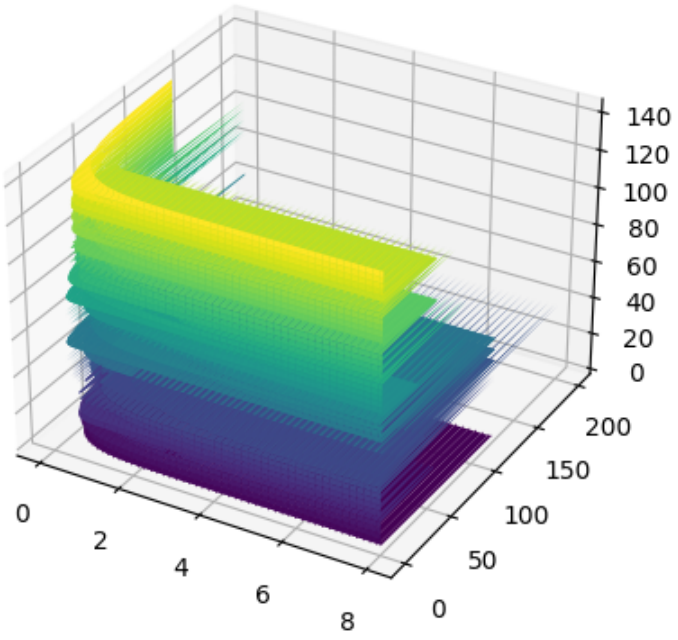


Figure 7.2: 3D plot of all cases, the “Work to be Done” response time histories

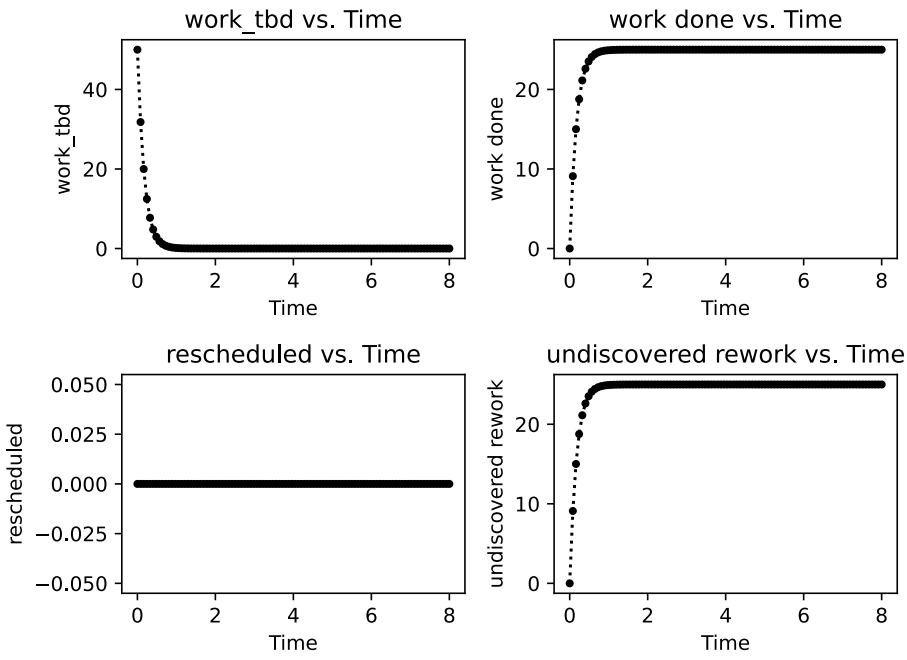


Figure 7.3: Case 100 Primary Stock Responses (Work To Be Done, Work Really Done, Undiscovered Rework, Known Rework)

be substantially more complex and thus falls outside the scope of this thesis.

7.4 Model Integration

Part of Experiment 1 involves investigating the possibilities to integrate the SD simulation with either a system model in SysML. This issue is addressed further below.

7.4.1 Possibility for Co-Simulation

There are many options available for integrated simulation of the SD model with other models. While Vensim does provide some interfaces to this end, the development work is perhaps a bit specialized. In this case, with a relatively simple model, the benefit is not necessarily present. Further, due to the restricted nature of Vensim, other tools like SD.js might be more beneficial for the graphical portrayal aspect of SD modeling and to leverage web technologies without additional licensing fees. However, those

tools may have more restricted co-simulation capability than commercial solutions.

7.4.2 Extension by Function-Passing

The Python implementation was kept at a simple level of complexity. However, various aspects of the Python implementation can be modified to enhance modularity. For example, in the base implementation here the interface is primarily in terms of the model parameter values. However, a more adaptive interface might allow the user to specify a function defining the rework discovery rate, or perhaps even the other individual rate functions. In principle, some advanced configuration at the level of Python functions is practically feasible with further development.

7.4.3 SysML Interface Possibilities

One of the final steps of this experiment is to consider routes for coupling the SD model with a system model. Assuming the system model is to be represented in SysML per Experiment 3b, then there are a few possible routes to obtain the desired functionality.

Represent Math in SysML using SysML-Modelica Transformation

In this case, a SysML representation would serve as a replacement for the SD models elsewhere (e.g. in Vensim, Python, etc). While unclear whether present Modelica libraries for SD modeling are support under transformation specifications, the SysML model might be co-opted to represent the flows or signals of the SD model. A representation of the flows via internal block diagram modeling (ports, interface blocks) coupled with parametric diagrams and constraint blocks is the target for example in NoMagic's SysPhS profile for Modelica transformation. If successful, this approach would supplant other analysis models by creating a new Modelica model for simulation on demand based on the content of the SysML model. After running the

Modelica model, the Systems Engineer could then take appropriate action. However, note that in this approach the modeling environment is now SysML directly, losing the Soft-Systems objectives claimed by Walworth et al[42] in using SD modeling with its graphical user interface.

Represent Math in SysML using Model+Simulator Formulation

Similarly, several SysML modeling tools are capable of simulating different parts of the SysML language. In this way it is feasible to create a simulator of model as well as the model itself, all in SysML as in Cole [110]. For this approach, the simulator would act as the integrator for the dynamics as represented in parametrics. One potential downside is that this direct approach may not provide the best numerical integration results, as the Systems Engineer would be responsible for implementing the integration scheme via the simulator model.

Develop Co-Simulation Capability to Vensim-like Tool

Tools such as Vensim may provide commercial co-simulation capability, such as FMI or other interfaces. These interfaces may require specific licenses. However, by using these interfaces, specifically FMI, it would be possible to run the SD model as an FMU. If this were feasible, a SD modeler could create the model directly using the graphical notation, and then export an FMU result for the Systems Engineer to track.

Create Mapping to Python Utility

Finally, the approach recommended here is to map some portion of the SysML model to an interface defined by the solver tool. In this case the target for integration is the Python code. A set of command line arguments can provide a simple interface accessible via scripting languages. This interface increases the flexibility of the Python SD modeling tool, as well as helps to generalize the technique by which the SD model

data is incorporated into the SysML model. It may also be possible, following the extension discussion, to pass some functional information as to how to relate the stocks (e.g. for rework discovery) given an appropriately defined interface and set of permissible/known functions.

7.5 Comparison to Requirement Status Data

In order to better understand the relevancy of the SD model from Walworth et al, and any associated lessons on parameterization, it is important to understand what requirements status measures might look like during the course of SE activities such as Requirements Validation.

7.5.1 Comparison to Friedenthal and Oster

The main issue in this experiment is the parameterization of requirements status. Unfortunately, it is not clear how exactly Walworth et al[42] would have intended their model's parameterization to relate to SE task measurement — or even if it ought to, given the soft-systems approach. Consider for example the canonical problem in this thesis taken as Friedenthal and Oster[98]. According to Friedenthal and Oster, the mission requirements come about by refining stakeholder requirements according to "analysis of mission use cases and scenarios"[98, p. 43]. Of key note for spacecraft design, "the orbit analysis is used to establish the Spacecraft orbit, which is used to derive many other system requirements"[98, p. 48]. In this sense, the canonical problem is presented as a fait-accomplis throughout the text — analyses are performed without much difficulty, measures of effectiveness and value are maximized separate from and subject to the requirements, and each step proceeds inevitably from the previous step. The interesting difficulty is that, regardless of any gaps or issues in the canonical problem, it is focused on the tasks which must be performed. In particular, it is not clear whatsoever that any factor such as "learning power" plays a role in

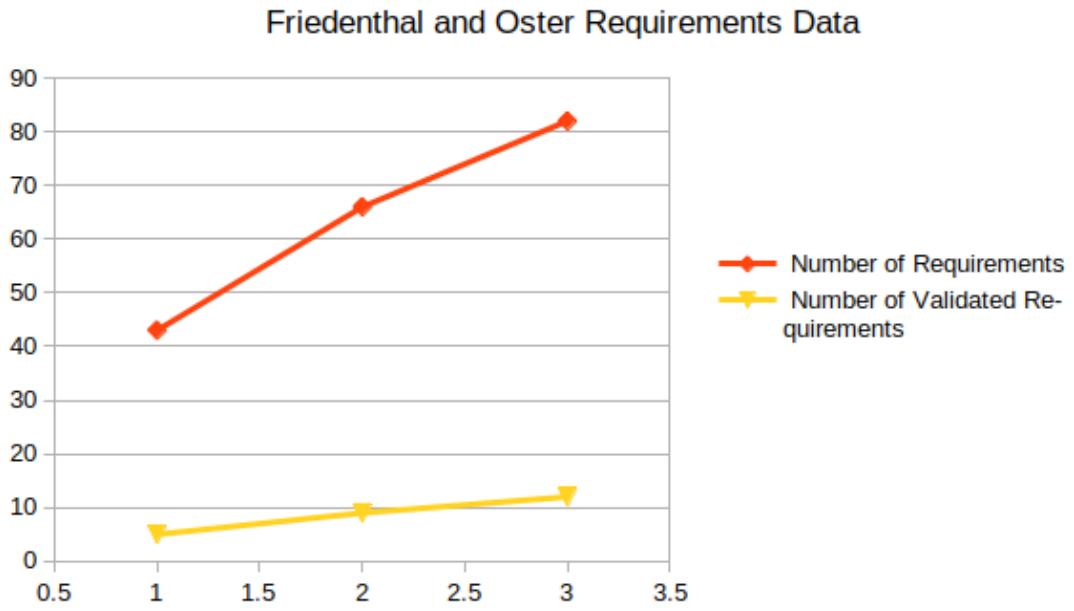


Figure 7.4: Counted requirements in Friedenthal and Oster[98] where "time" corresponds to major milestones, such as receipt of stakeholder requirements, system requirements analysis, and payload requirements analysis.

orbit analysis. The result is that despite any apparent similarities in work really done according to Figure 7.4, it cannot be said whether Walworth et al is predictive. Walworth et al may be merely an explanation among many for the appearance of s-curves in various measures of SE performance. Consider additionally that the total number of requirements (potentially a proxy for "total number of tasks" for the activity of requirements validation) is not known until the end of the requirements processes. The total number of requirements which must be validated is not generally known before commencing SE activity. It may be possible to estimate the final number of requirements based on some measures of the problem or system use cases or functionality, perhaps even according to system similarity to previous experience; however, no such estimation is performed in the canonical example.

7.5.2 Comparison to Anonymized Requirement Data

Beyond the canonical problem and its potential flaws, Walworth et al express the desire to track "the technical maturity of projects" through "planned parameter profiles"[42]. Under the interpretation of a planned parameter profile, the Walworth et al model has application well beyond requirements validation, and the overarching hypothesis of the Walworth et al model is its applicability to any process which is modeled in analogy to the learning system. Thus far, the main concern in adopting this model has been the validity of the parameterization and the ability to adopt and map parameters to responses.

Perhaps data can illuminate the picture — no real data is presented in calibration with the Walworth et al model, as it is justified on the basis of Subject Matter Expertise according to the Soft System Methodology[42]. In Spring 2020, a study was performed to revise a set of requirements in order to improve the quality of a requirements database for a commercial aircraft development project. This study included requirement text, but also requirement relations to various SE artifacts to better enumerate items which have traceability to the requirements database. These relations include traceability to analyses in the explanatory sense, as in Friedenthal and Oster[98]. Over the course of 64 days, four (4) engineers modified requirements according to the following procedure, which is a summary of communication from this present author to the requirements team on March 3, 2020.

Requirements Modification Procedure

Step 1: Prepare Create requirement review tracking data (definition of personnel)

Step 2: Begin Select a requirement for review by applying the appropriate labels (stereotype). In the SysML model, this was captured via a Stereotype «Reviewed» with properties for Date, Notes, and ReviewedBy. These properties

manifest as tags on the requirements to be filled throughout these steps. Assignment of a requirement was to be indicated by filling in personnel details under ReviewedBy, while the date was reserved for the last date on which the requirement was modified.

Step 3: View Create a view on which the requirement, including its text and all relations (at least a depth of 1 depending on the complexity of the relationship network), so that they may be inspected

Step 4: Text and Name Inspect the requirement text and name. Text should have active voice and no grammatical mistakes. Conformance with requirement templates is optional but preferred.

Step 5: Requirement Relations Step 5 is the most constrained by the MBSE language, which in this case was SysML. In Step 5, engineers should inspect the requirement relations. First, check that relations are applied correctly. Then, investigate whether important relations are missing. The nouns of the rewritten requirement text may be particularly helpful guidance here. If an important entity is missing from the SysML model or has a different name, take appropriate actions to the System Model or Requirement in order to establish the necessary relation. Note: this step may induce changes to the system architecture beyond requirement modification; however, no major system architecture revisions were performed and such revisions primarily amounted to adding or renaming values in various locations, alongside requirement-architecture traceability enhancements.

Step 6: Documentation In Step 6 the engineer should provide some documentation about the changes made in the appropriate field of the revisions label (stereotype tag). If all modifications are completed, the Date field is to be marked to indicate completion of modifications.

Additional procedures were also given for test case definition, however these were not tracked in a measurable sense to the extent of the requirements themselves. As described in the procedure, the result includes a dataset of requirements according to date modified (or date by which modification was completed), notes, and the modification author.

Summary of Requirement Modification Results

An initial set of approximately 55 requirements targeted for revision grew to 125 in the final count. Some caveats apply to this number, which will be discussed shortly with respect to visualization of the modification data. Revision began on February 20, 2020 and concluded on April 24, 2020. Engineers are anonymized in this data as E1, E2, E3, and E4. As far as experience working with requirements, in a non-linear ranking the individuals would be listed as E1, E2, E4, and E3 according to the relative level of experience. However, the difference in experience between each individual is not evenly distributed, nor is experience measured directly in this study. E1 and E3 began the effort on February 20 with a split of the initial requirements set, while E2 and E4 joined later in the course of the revision effort. While the data do not disprove any assumptions about the s-curve applicability to a parameter profile, they do call into question the parameterization suggested by Walworth et al and in general for requirements status.

The "final" history of requirement revision is presented in Figure 7.5. Unfortunately this data is not straightforward. In particular, E2 revised many requirements as captured in the revision history of the system model, yet after the modification period removed many of the revision labels. The overall numbers for E2-1 and E2-2 add up to the final tally for E2 as presented in Figure 7.6. Figure 7.6 gives the distribution of final requirements modifications per engineer. While not an exponential model in terms of the number of requirements, this model may still conform to the ideal of the

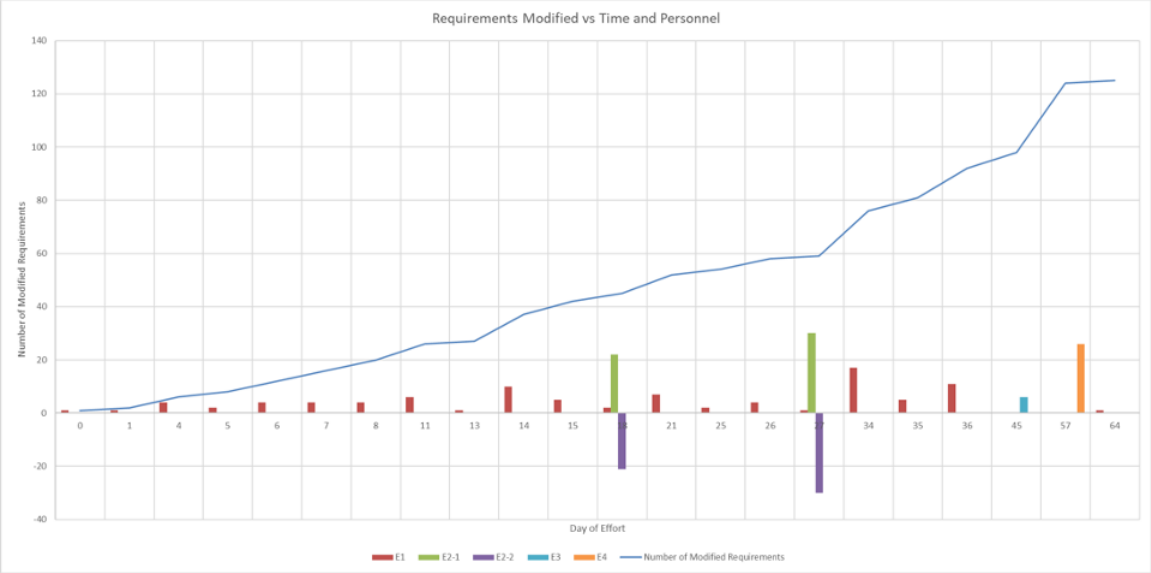


Figure 7.5: Requirements modified vs time per engineer during requirement revision activities

s-curve but portray the state of the measure before the inflection point. It might be argued that the learning curve is visible for E1 from $T = 0$ to $T = 14$, yet E3 only completed 6 requirements and submitted all at the same time, with 12 of the original assignment having been shifted. Additionally, E4 submitted all modifications simultaneously, but these modifications were primarily in authoring new requirements into the database rather than affecting some improvement from an initial state — E4's requirements were all new. E2 did make some real modifications; however due to most of these modifications being to name or requirement ID rather than relations or text, it is assumed that E2 removed labels out of concern for indicating that revision was completed. However, more important for this experiment is how the variables which are measured differ from the assumptions under Walworth. First of all, not all personnel work from day 1 and not all personnel work the same. The number of requirements (work to be done) is not fixed from day 1 and increases throughout the effort. In fact, at least 10 requirements at the end of the period were marked for revision but not yet modified. Furthermore, the intensity if measured as requirements per engineer - day is not constant and fluctuates. The difficulty becomes that even if

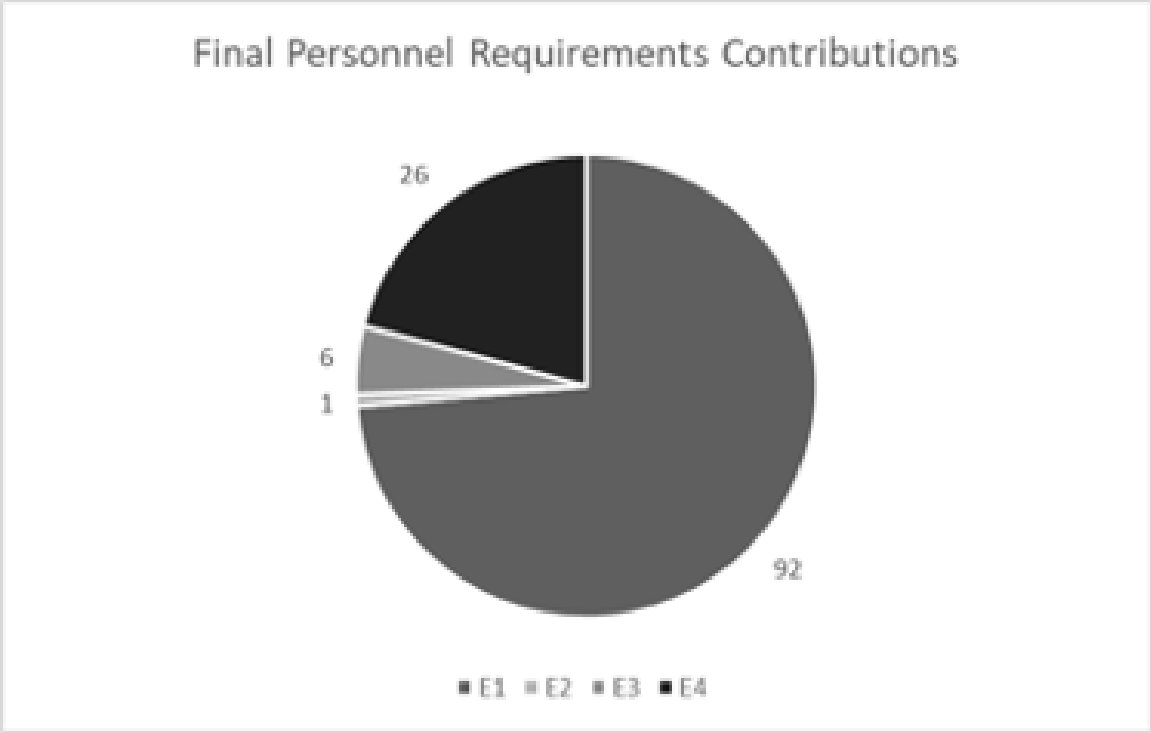


Figure 7.6: Final requirements modified per engineer status

the Walworth et al model is valid, values for the parameters such as learning power and intensity cannot be chosen appropriately for planning purposes, especially since historical data was ignored by Walworth et al in particular for the noisiness described above[42].

Additionally, consider whether the measurements in Figure 7.5 indicate an improvement. In all, tracking modification does not measure improvement in the understanding of or quality of the requirements database. For example, the most that can be said here is that the 125 modified requirements are more closely in compliance with the ad hoc standards expressed by the procedure after modification than before. However, some other measures would be necessary to indicate improvement. Before modification, stakeholders considered the requirements primarily as necessary in existence but not important otherwise due to interest in high-fidelity analysis; afterwards, this is still the case, and the long-term value of working these improvements with respect to these stakeholders is unclear. Measuring improvement might

also require definition of program-specific events, such as an agreed demarcation by which the requirements engineer might declare a requirement validated, or to have improved in validation status. Furthermore, due to the nature of an MBSE language such as SysML, measuring the improvement of requirements understanding through requirements modification may also necessitate architecture measures, as elements and relations must evolve to express the traceability of the requirements co-equal to the requirement text. However, the degree to which this co-evolution is necessary should be considered program-dependent and may not be absolute — an MBSE practitioner may desire some amount of traceability to say that a requirement specification is understood, but how much traceability is actually necessary to understand requirements is not clear. Finally, it is worth revisiting the requirements status models and in particular Grenn et al[43] from Baseline C2. Grenn et al's model provides a probabilistic forecast of requirements quality and the person-hour effort required to obtain a desired level of quality. The key issue in applying this model, other than patent licensing, would be in properly defining the 14 quality attributes which Grenn et al establish for requirements. In particular, given the concerns which have been raised here regarding the definition of parameters in the Walworth et al model, two attributes stand out from Grenn et al as particularly difficult: correctness and design independence. Necessary to facilitate the application of this model would be appropriately strict definitions of these attributes and a system for applying them to requirements, such that the current state of the entropy measure is known and can be used for forecasting purposes. Correctness and Design Independence seem likely to be controversial in definition and application due to the subjective nature of system architecting, and though organizational or program standards might ameliorate this problem, to an extent such standards shift the problem of agreeing on sufficiently precise definitions to those who author the standards, and trust that their authority will be recognized, and leave the potential for measurement issues arising from iterative

redefinition of criteria during a program.

7.6 Directly Representing Requirements Status

Based on the literature for requirements status [35, 15, 44, 41], the requirements status is frequently measured according to a set of counts, percents, as well as nominal and ordinal measures. For example, [41] indicate requirement attributes for requirement trends including "Process Phases, Disposition Action, Maturity States, Priority Levels, Cause, Impact Level, Classification Type, and Dates & Times" (p. 17); for validation and ensuring user needs are met, "Maturity State... Stakeholders... Architecture Level... Process Phases, Disposition Action, Priority Levels, Cause, Impact Level, Classification Type, and Dates & Times" (p. 33); finally also for verification the same attributes are again relevant (p. 37) in addition, of course, to a yes/no validated or verified indication per requirement "at each level of the system development" (again, from [41]). While some attributes such as priority may be used for planning activities, typically an item of interest in requirement status is the definition of maturity states and how these states evolve with requirement revisions. After all, this portrayal of requirement measures is what motivates Grenn et al's [43] thermodynamic metaphor. A key paper in this discussion is Jackson et al[70], in trying to expose hidden aspects of SE and MBSE regarding requirements maturity tracking and measurement. This paper makes clear the discrete and event-driven nature of the requirements status vis-à-vis requirements maturity as a finite state machine describing the elevation of requirement maturity.

In Figure 7.7, a basic requirements representation is established with a single "attribute" — see [41] for additional attribute possibilities — which gives maturity as a set of possible enumerated values. These values appear nominal, but are intended in an ordinal sense (progressively greater maturity) and this can be indicated both via semantics as well as additional labeling on the model (e.g. tags indicating an integer

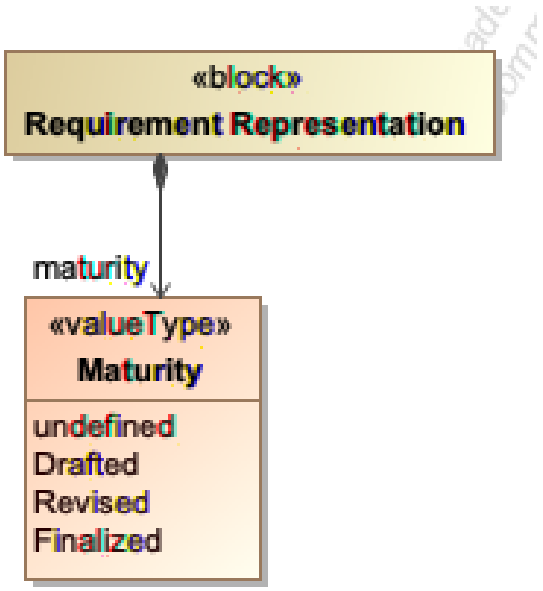


Figure 7.7: Simple Requirements Maturity Representation based on [70]

value corresponding to the enumerate). As far as semantics, Figure 7.8 presents a state machine for the evolution of a requirement. In particular, the idea is that a requirement's maturity continues to evolve according to some discrete events which trigger, establish, or assert a new maturity state — or removal of the requirement, as discussed by Grady [15] in requirements validation and Figure 3.2. This state machine behavior gives a simplified meaning to the evolution of the requirement maturity attribute, simplifying the behavior described by Figure 2 in Jackson et al, while mirroring the general structure of Jackson et al's Figure 6[70].

This discussion comes to two points. Firstly, regarding requirements status parameterization and Hypothesis 1a, there are more questions than answers. The attributes of requirements described by Roedler et al in the Leading Indicator guide[41] are not precise; the lack of precision is revealed by Jackson et al through clarity about how a requirement might become more mature. That is, according to Jackson et al, a requirement elevates from "Candidate" to "Provisional" status on assertion by an author, while the Requirement elevates from "Provisional" to "Reconciliation & Negotiation" when specific teams concur on whether acceptance criteria are met[70].

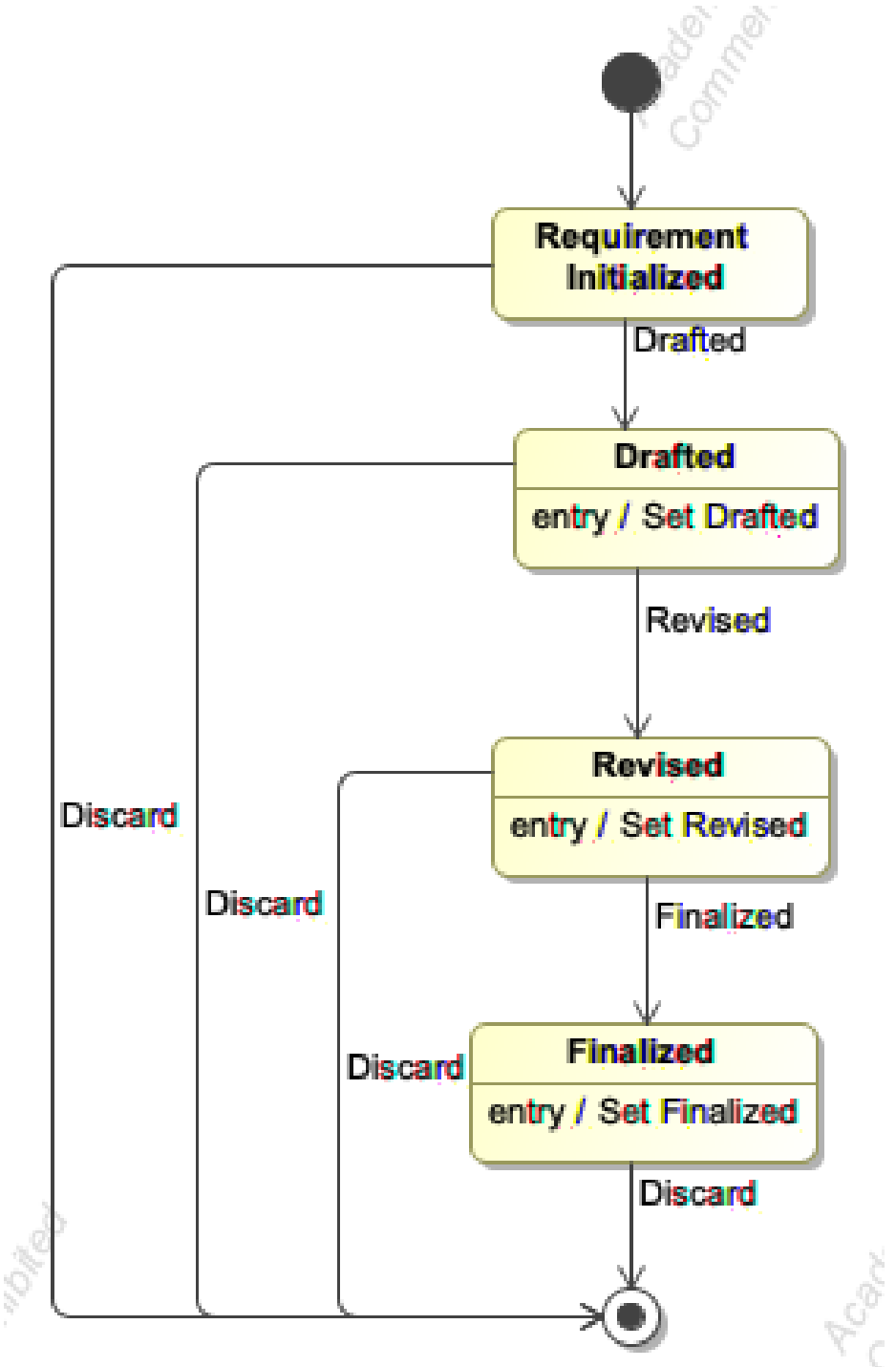


Figure 7.8: Simplified state machine for requirements evolution based on [70]

These events are more qualitative in nature than a quantitative view which might be seen as preferable from a background in physics-based modeling, traditional aerospace disciplines, and the thermodynamical metaphor presented by Grenn et al[43] where

requirements status or maturity elevates according to quantified parameters. Likewise, even the imprecise parameters of Walworth et al were quantitative in nature. The key issue is that practical requirement status parameterization is *(*event-driven), perhaps to the detriment of Morecroft's [107] argument, as organizations assert a process by which engineers will sign off on the formal requirements specification document. Thus, the second point to address here is the appropriateness of the baseline models which demand quantitative parameterization. Instead, perhaps a discrete-event simulation may more closely mirror the inherent structure of organizational processes as well as requirements status/maturity evolution. This latter point is to be investigated further in Experiment 2.

7.7 Recommendations and Conclusion

Experiment 1 can be characterized by the emotion of disappointment. In all, rejection of the Hypothesis 1a is probable. The main cause for rejecting Hypothesis 1a is the apparent mismatch between the capability and performance of the parameterized numerical models for requirements status, vs. actual practical descriptions of requirements status. Key issues arose in replicating Walworth et al due to the lack of an appropriately parameterized rework discovery "exposure rate"[108]. Despite these issues, a simple model can still be constructed by assuming some proportionality relation for the discovery rate. The simplified model lacks the jagged features of Walworth et al due to the non-existence of probabilistic effects. To this end, Experiment 2 may provide greater insight and promise towards the overall modeling effort: either by providing a rate calculation for rework discovery or by replacing the relation to work really done. Perhaps most important, the SD model(s) may remain critically important — not because of their parameterization or simulation capability, but because of the narrative that their visual portrayal can tell. This narrative capability and the motivating soft systems methodology may be the primary benefit of

Walworth et al, although Walworth et al do not go nearly as far as Lyneis 2007[111] in specifying extended causal loops for the learning system. Perhaps Experiment 1 has been trapped by Lyneis 2007's "hopelessness" feedback loop added to the learning system archetype. While there are certain interesting questions — how does team experience and capability affect the time elapsed between requirement maturity elevation events, for example — there does not yet appear any means by which to answer these questions.

8

EXPERIMENT 2: DEVELOPMENT OF METHOD MODELS

8.1 Recap of the Experiment

The experimental procedure in response to Hypothesis 2 and Hypothesis 3 is to formulate simulation models firstly for known validation methods, such as Grady's textbook diagram from Figure 3.2. These flowchart descriptions might normally be simulated by DES. The first step will be to construct a model of each kind. Then, the model must be run to examine parameter variation. Repeated simulation will be leveraged to produce result distributions to see if the behavior and performance of the models is similar. This procedure is summarized by the statement in Experiment 2.

The planned experimental procedure was described as follows:

- Construct DES (e.g. PyDES)
- Simulate DES with some parameter variation to see its impact
- Check for compatibility regarding the System Dynamics model
- Plan a mapping of the DES model to a system model

Hypothesis 2 is defined as:

If SE methods are also processes per Martin[4], then they can be simulated by process models like DES or ABS.

Accordingly, Hypothesis 2 is supported if the simulation of the SE methods can be achieved by DES. Likewise, Hypothesis 3 states:

If SE methods are treated like other business processes, then bi-level hybrid simulation will provide better parameterization for task-planning purposes.

Thus, due to the apparent advantages of bi-level hybrid simulations, the creation of a DES for the SE methods should somehow improve parameterization of the methods for task-planning purposes.

There are clear shortcomings in these hypotheses. In the first case, falsifiability is questionable. In the latter case, this experiment only begins to provide an answer. However, consider the approach taken regarding Experiment 1. In developing a model, each assumption is in turn a form of hypothesis supported or rejected based on the success of the prototype towards its objective. This vein of practical reasoning is again to be applied for the present experiment. While Hypothesis 2 may appear obvious or a foregone conclusion — that is, given sufficient effort — note that there exists a disconnect between the reasoning of Walworth et al [42] used in Experiment 1 and the assumption asserted by Hypothesis 2. The disconnect is that Walworth et al do not believe that DES is capable of capturing the evolution of requirements status appropriately as a "functionalist representation"[42]. More precisely, Walworth et al sought to create a computational model representing the evolution of requirements status with relevant parameters without accounting for the steps through which the project/status must evolve — see discussion in Experiment 1 involving [70] — whereas Experiment 2 intends to directly consider methodological proposals and in particular proposals regarding requirements validation or quantification activities. First, the canonical methods for comparison will be given further definition.

8.2 Method Definitions

There are several important factors for consideration in Experiment 2. First and foremost are the Systems Engineering methods under consideration. While the objective of this thesis is not to say with any conclusiveness that one particular SE methodology

is better than another, direct comparison of SE methodologies is necessary in order to explore decisions in Systems Engineering planning and step in the direction of aiding decisions regarding specific tasks. For this purpose, two Systems Engineering methodologies have been selected:

- Simplified OOSEM in Architecting Spacecraft (OOSEM-lite)[98]
- State Analysis (SA)[21, 112, 113]

Additionally, [91] provides an outline of a product development process for the canonical system (FireSat) and that will be outlined here.

8.2.1 Overview of OOSEM-Lite

The Systems Engineering methodology in Architecting Spacecraft[98] is a simplified version of the Object-Oriented Systems Engineering Methodology (OOSEM) presented in earlier works by Friedenthal[114]. In the newer book from which the canonical problem is derived, the primary set of tasks involves the "Mission & System Specification and Design Process"[98]. The overview of the design process is detailed in the text and available online (e.g. [115]). The overall process is depicted in Figure 8.1, parts of which are governed by the license associated with [115]; to this end, a stereotype indicating which aspects are modified from the original IP is used to indicate where modifications have been made.

Figure 8.1 shows a series of steps from the very beginning of the system lifecycle through to requirements verification and presumably delivery of the spacecraft to a customer. As such, while a partial lifecycle process, it covers most requirements-related activities in the development of the canonical system, FireSat. Two activities which are the primary focus of the present study are "Analyze Mission & Stakeholder Needs" as well as "Specify System Requirements", as these activities are explicitly about generating, modifying, and finalizing requirements according to the text[98].

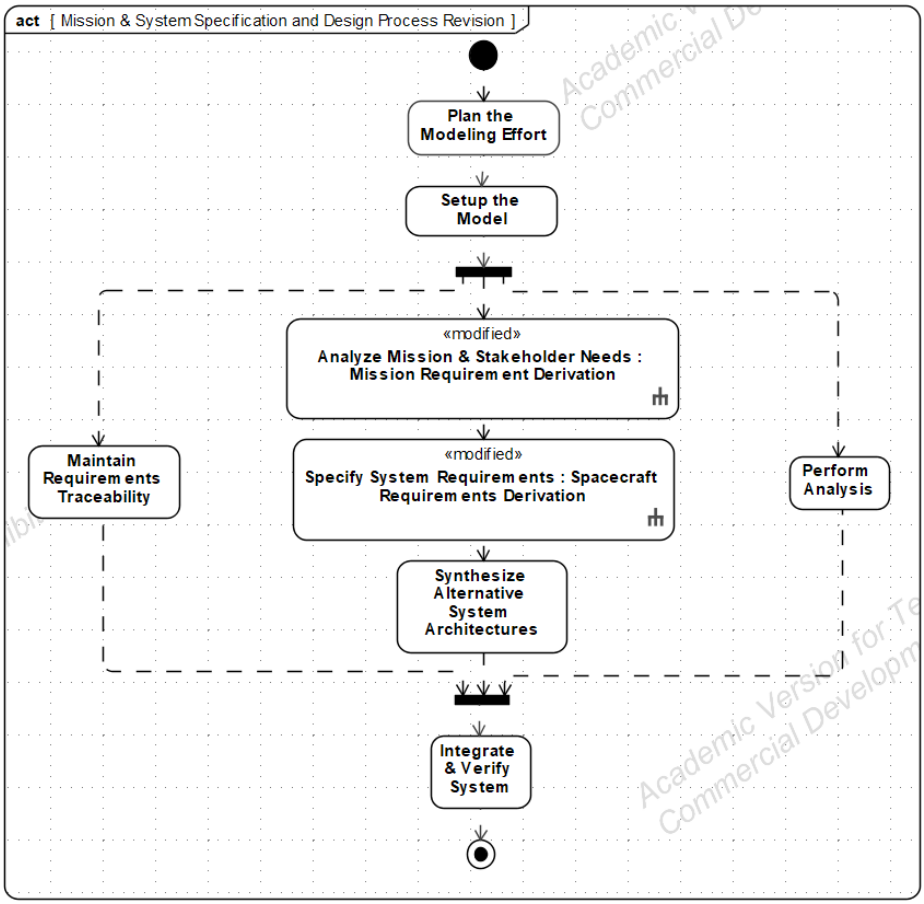


Figure 8.1: OOSEM-based lifecycle process from Architecting Spacecraft[98, 115]

A later step to "Synthesize Alternative System Architectures" is used in the book to generate and modify subsystem requirements; however, as some subsystems have a greater presence in the previous step of System Requirements, the main intention of the synthesis phase appears to be analysis and selecting discrete options for certain complicated spacecraft systems, such as electrical power. Mission Requirement Derivation is illustrated by Figure 8.2 while the lower-level Spacecraft Requirements Derivation is given in Figure 8.3. Both Figure 8.2 as well as Figure 8.3 were constructed according to the text and were not given in the original SysML source [115].

Figure 8.2 provides detail on the Mission Requirements Derivation tasks (the method for the process) involved to specify mission requirements. In the context of FireSat, mission requirements are primarily requirements which relate directly to

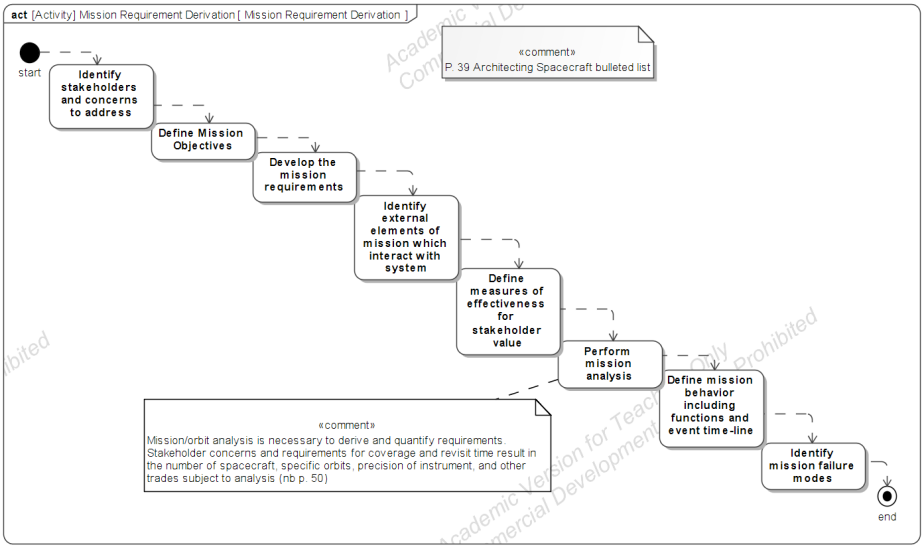


Figure 8.2: Mission Requirement Derivation from Architecting Spacecraft[98, 115]

stakeholder objectives and should specify the orbit of the spacecraft. The spacecraft orbit aims to directly meed stakeholder needs, especially for Earth observation, as it drives the timing and quality (or imaging technology requirements) which can be expected. Moreover, it is important to note that the spacecraft mission inherently involves multiple other systems: ground systems, other spacecraft, end users which may send or receive data or commands to ground stations and/or spacecraft, launch vehicles, etc. The mission objectives phase may involve some specification of these other ancillary systems at a high-level and their definition may also be pursued if they are not to be selected off-the-shelf. For Friedenthal and Oster[98], the primary concern is the spacecraft under development, and this is the focus for further refinement after solidifying the mission. The further refinement is illustrated by Figure 8.3, whereby the tasks are given which determine the spacecraft system requirements given a mission definition. Additionally, an overview of the quantity of requirements specified over the course of these steps was previously illustrated in Figure 7.4 from Experiment 1.

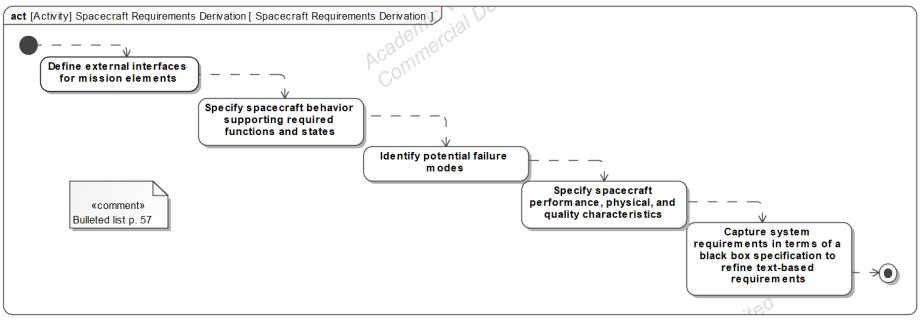


Figure 8.3: Spacecraft Requirements Derivation from Architecting Spacecraft[98, 115]

8.2.2 FireSat Product Development Process

FireSat is a canonical system from the spacecraft design community which has seen several iterations in text. *Space Mission Engineering: The New SMAD* (New SMAD) introduces a process for FireSat called "Space Mission Engineering [which] is the refinement of requirements and definition of mission parameters to meet the broad objectives of a space mission in a timely manner at minimum cost and risk" (p. 45)[91]. Space Mission Engineering (SME) is distinguished from a SE process, insofar as the emphasis is on "whether the needs of the end user are satisfied," (p. 45) which appears to be in line with product development processes. Figure 8.4 provides a drawing of Table 3-1 from [91] which outlines this product development process.

Many of the steps in Figure 8.4 overlap with the Mission Requirements Derivation and Spacecraft Requirements Derivation activities from Friedenthal and Oster. Wertz defines this process from Figure 8.4 as "needs-based" rather than "capability-based"[91]. As such, these two steps in particular are considered to cover most of the New SMAD concerns while including the iterative requirements development process which is affected to an extent by validation concepts, despite lacking iteration.

8.2.3 Overview of State Analysis

State Analysis (SA) is a SE methodology which is inspired by control theory and the the application of the archetype of Plant-Estimator-Controller (designing both the

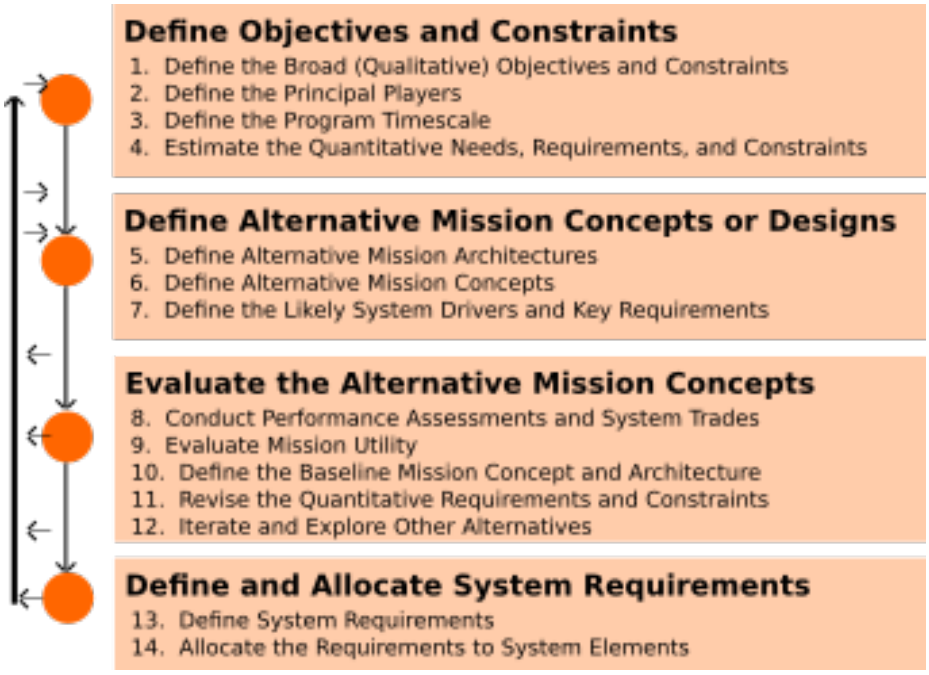


Figure 8.4: The Space Mission Engineering Process (Table 3-1) redrawn from [91]

control system and system under control) in all aspects of the system development process/lifecycle[21, 113, 112]. Kordon et al[113] provides detail on how to synthesize traditional functional analysis with SA for JPL project formulation. Additionally, Kordon et al emphasize the need for inter-process communication over event handling in the SE process[113]. Overall, SA consists of three main phases[113]:

- First aspect of SA: state-based behavioral modeling
- Second aspect of SA: goal-directed operations engineering
- Third aspect of SA: state-based software engineering (exercise and execute the model[21])

A SysML activity diagram illustrating the overall process is illustrated by Figure 8.5. Kordon et al specifically compare SA to Functional Analysis (aka Structured Analysis from Grady[11]) in that while Functional Analysis can be adapted to model-based SE tools and environments, it lacks representation of dynamic behavior and interactions, i.e. emergence, apparent in complex systems — be they highly automated vehicles

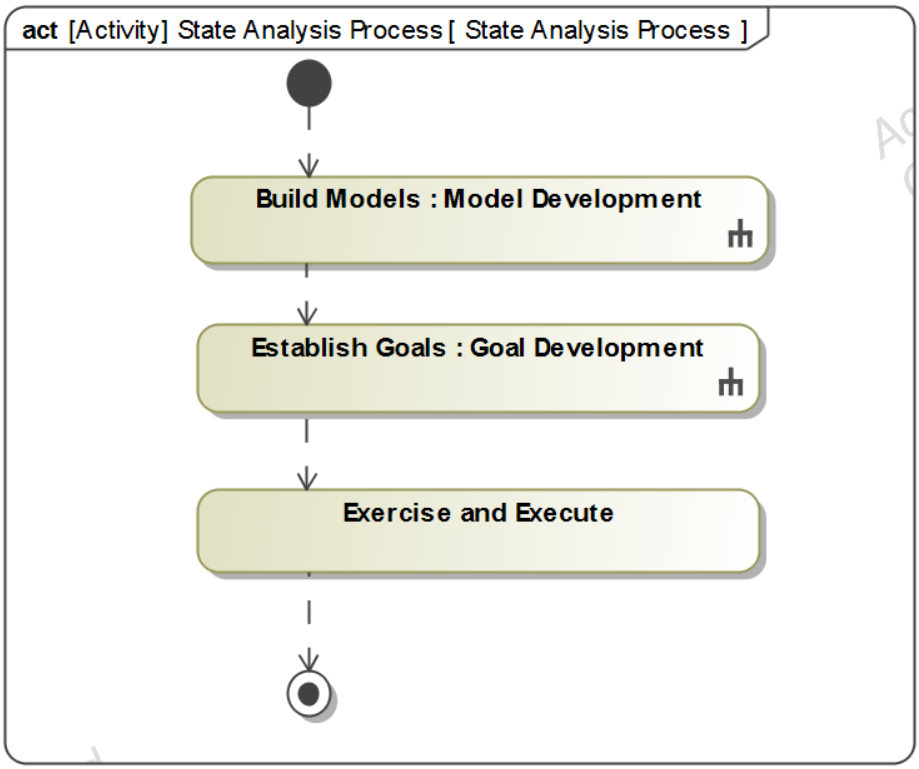


Figure 8.5: Overview of the SA process.

or cyber-physical systems, or interoperating vehicles and infrastructure with common purpose[113]. In order to tame this complexity, Kordon and others propose the steps listed above. Specifically, in early development the equivalent of the architecture specification through Functional Analysis is seen as the state-based behavioral modeling[113]. The behavioral modeling step hinges on the definition of state variables and state models, as illustrated by Figure 8.6. The relevant comparison is in terms of how this process bears any similarity to OOSEM-lite Mission Requirements Derivation and Spacecraft Requirements Derivation. In the case of the canonical spacecraft, as discussed, the SE methodology appears to assume that all activities proceed forward in an orderly manner. Yet, at each step, various mathematical models for mission design or system performance measures are applied in quantifying spacecraft performance measures and setting new requirements. These models must be constructed, and the new combined model for the particular analysis case which is

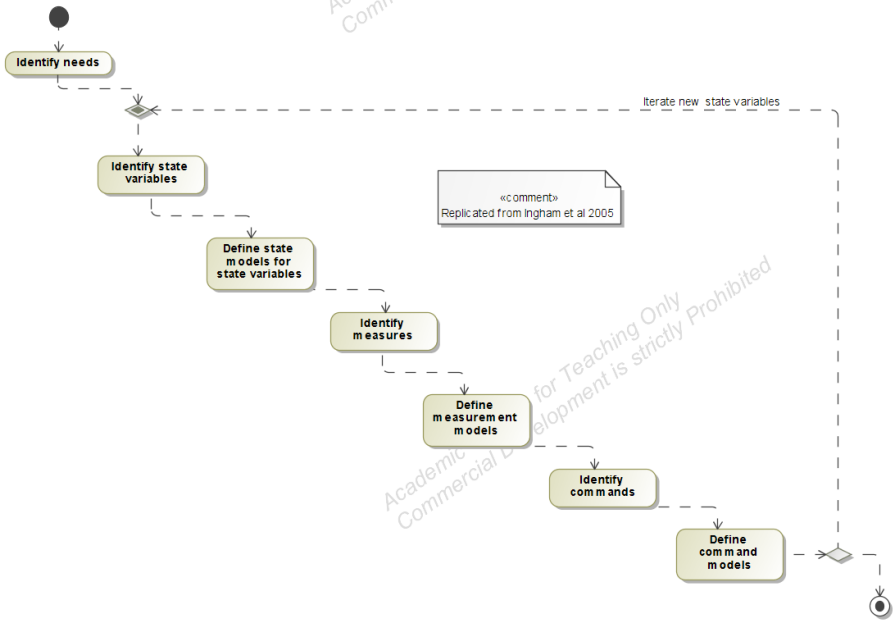


Figure 8.6: Model Development process described by Ingham et al[21].

connecting each aspect for the performance calculation must be shown to be valid per earlier discussion on validation. Therefore, the canonical description obscures the real model development process which may be required in spacecraft design. Even though SA is focused on "state" and the control aspects, it is inherently about constructing models. Thus, consider that the iterative nature of model development would apply also to the OOSEM-lite steps if new models (e.g. especially for mission design, or system technologies, or complex mission architectures) were needed in the assessment of requirements and system functionality or suitability. When the SA model development process is complete, then all state variables have been defined and all state, measurement, command models have been defined. This step is iterative in terms of state variables uncovered in the model development process for control. The next phase, Goal Development, may involve iteration back to model development per Figure 8.7. In the SA Goal Development phase, the SE enterprise is setting sequences of commands to achieve objectives with respect to performance quantities measured by the estimator models and aligning these sequences on an overall timeline. Impor-

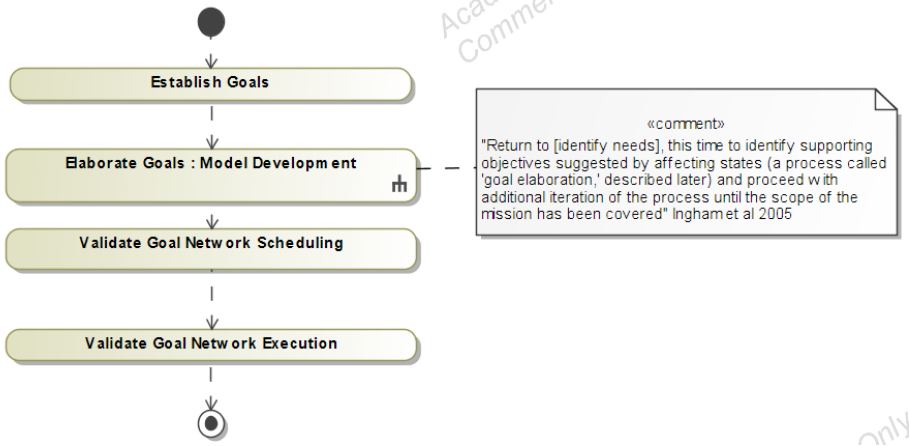


Figure 8.7: Goal Development Process extracted from Ingham et al[21].

tantly, Goal Elaboration is a process by which additional necessary steps in achieving a goal are broken down and enumerated, and this may result in the identification of new state variables triggering model development iteration.

Finally, the resulting model is to be used in the development of hardware and especially software. The model itself may be suitably *executable* in the sense that it can be simulated or evaluated against a decision problem — useful not only as a testbed, but even as model-based software in real systems, according to Ingham et al[21]. Clearly, SA has in mind systems amenable to a high degree of automation and computer control, as befitting its origin at the Jet Propulsion Laboratory. In particular, Castet et al 2015 illustrate a full implementation of a SA inspired system design for a flashlight and spacecraft (though not strictly using SA, per se, rather concepts inspired from it)[22]. Other MBSE efforts have also applied SA directly, such as the European Extremely Large Telescope (as reported in Wagner et al)[112].

8.2.4 Validation Processes

Of specific interest are tasks within these SE methodologies in which validation actions occur, specifically with respect to requirements in early phases of development. Typically, these may be seen during analysis as described earlier in Chapter 3. The

relevant tasks are the specification of mission and system requirements for Simplified OOSEM, and the model development process for State Analysis. For the selected tasks, a method of simulation must be selected in order to help determine the necessary means of expression of the tasks for simulation. However, before beginning to dig deeper into these SE methodologies, it is important to consider the general problem. As discussed in Chapter 3, Grady specifies a validation process which involves requirements evaluation, whereby requirements might be added, modified, or deleted, by potentially identifying new requirements, rewriting existing requirements, or discarding with the aim of determining completeness and correctness of the requirement set specifically at what Grady terms the "item" level[35]. How requirements might be analyzed, assessed, or rewritten has substantial leeway. For Grady, requirements must be quantified and "We can gain confidence in requirements values that were derived from a system math model for the parameter in question because it enforces discipline on value assignment"[35, p. 684]. Beyond numerical issues in requirements, the wording may also be incorrect: as such Grady describes by item requirement validation a process of error detection and error correction [35, Note "Content in error" Figure 8.18, p. 685]. Therefore, before investigating processes which involve validation further, the process of error detection should be considered first as this process may govern the work more generally than any individual prescribed SE methodology.

8.2.5 Error Detection as Proxy for Validation

Consider a simple process by which a team accomplishes tasks. The basis for this discussion is Figure 8.8, which illustrates a conceptual view of the process.

Figure 8.8 presents four main phases. Overall this procedure includes a stack of tasks, a procedure or duration during which the task is completed, a review of the task, and either a reset to begin again or removal of the task on acceptance of the work. Part 1 of Figure 8.8 shows task 1 entering the process from the stack. Part 2

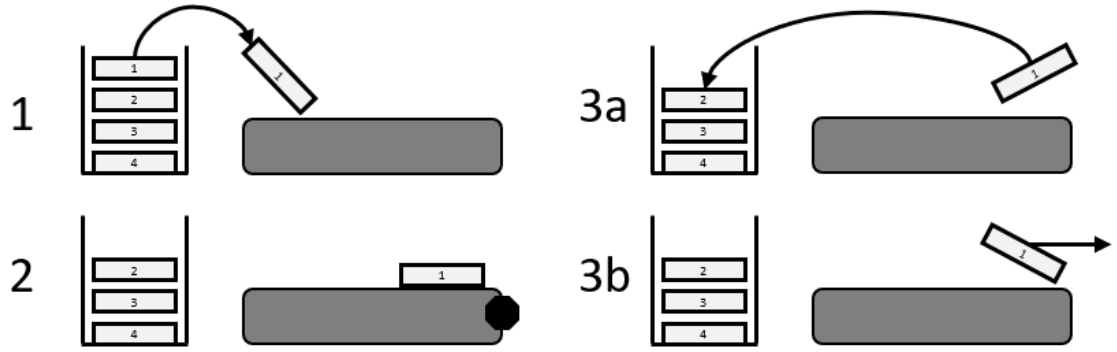


Figure 8.8: Simplified Error Detection Process during Task Workflow.

shows the task reaching the review stage. Part 3a shows the case where the work fails review — i.e., an error is detected — and the task is set back on top of the stack. Part 3b shows the task being accepted as complete and removed.

This conceptual model is simple. There are nuances which are not captured here, such as errors which remain undiscovered in the vein of Walworth et al. Additionally, tasks are assumed to be reworked in entirety, rather than in some fraction of the work in which the error exists. Tasks are processed one at a time. Under these simplifications, the stack of tasks can be represented by a monotonically decreasing positive integer value $t \in \mathbb{Z}^+$, the number of errors as a monotonically increasing positive integer $e \in \mathbb{Z}^+$, the working of the task as a duration with upper and lower bounds defined as $d_{pt} \in [2, 4]\,\mathrm{h}$, the probability of error detection p_{error}, the resetting and recording of the error as a duration with lower and upper bounds defined as $d_{eh} \in [5, 12]\,\mathrm{h}$, and the finalization of tasks as a duration with lower and upper bounds defined as $d_{tf} \in [30, 90]\,\mathrm{m}$. In English, that is that the model begins with some number of tasks, say 150, and then each task is processed for 2 hours to 4 hours. The precise duration is left to a probability distribution. Next, any error might be detected with some probability. If the error is detected, the activities associated with resetting the task take between 5 hours and 12 hours again made precise according to the same kind of distribution as the task processing. During the reset, the number of errors is incremented by one. If the task however should be finalized instead of

reset, then the task finalization takes between 30 minutes and 90 minutes. After finalization, the number of tasks decreases by one and the work proceeds until no tasks remain.

If a SE planner wishes to make use of this conceptual model in an MBSE environment, tools already exist for the purpose. Figure 8.9 provides a translation of the conceptual model into SysML as a pure activity model. This Figure is a direct translation of the statements before. To better understand the model structure, it is

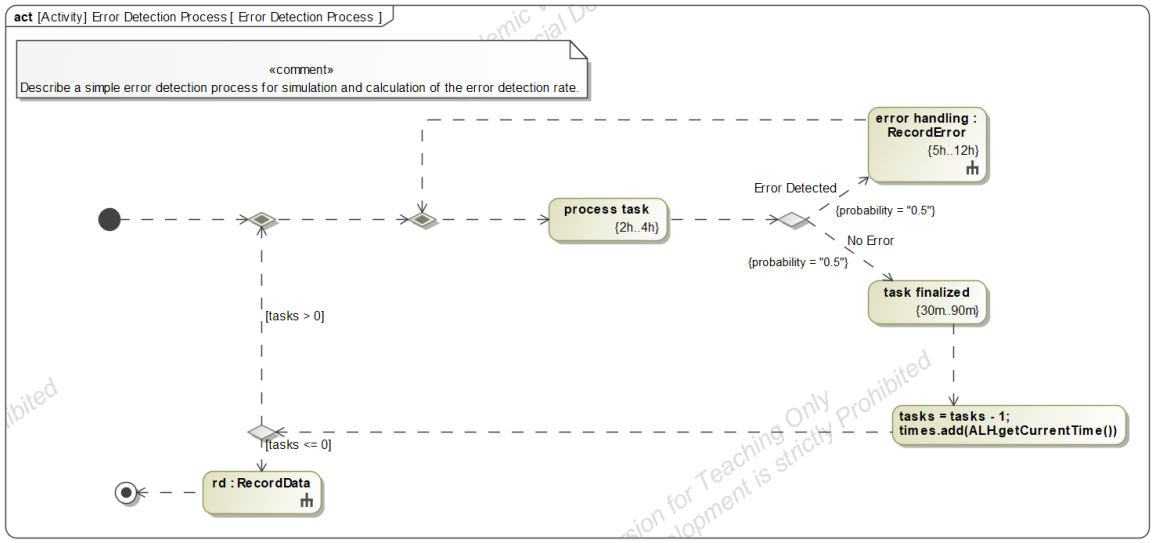


Figure 8.9: SysML model describing the error detection process from Figure 8.8.

possible also to generate a decompositional view. In Figure 8.10, the decomposition of the Error Detection Process is illustrated on a UML Class Diagram. Missing of

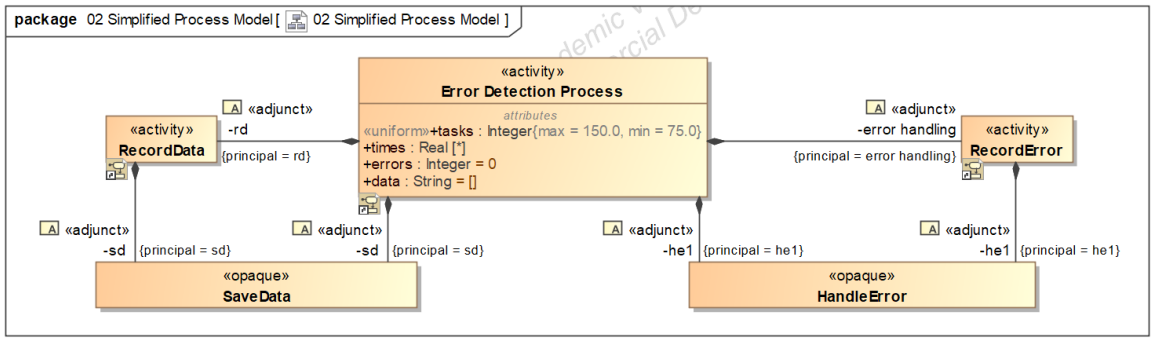


Figure 8.10: Decomposition of the Process showing the subprocess and values associated with the simulation.

course from the class diagram view are the values for the durations which may be interesting to change from one run to the next. While these values can be updated from Figure 8.9, it is also possible to provide a means of updating them via tables as in Figure 8.11.

Criteria

Element Type: Call Behavior Action ... Scope (optional): Error Detection Process ... Filter: \d

#	Name	Applied Constraints	Constraint Minimum	Constraint Maximum
1	error handling	error handling bounds=5h..12h	5h	12h
3	process task	process duration bounds=2h..4h	2h	4h
6	task finalized	finalization bounds=30m..90m	30m	90m

#	Name	Probability
1	Error Detected	0.5
2	No Error	0.5

Figure 8.11: Table for Durations and Table for Error Probabilities (sum to one).

By running this simulation, three files are generated. One JSON file with the simulation data, one CSV file in the standard format representing the error detection data, and one csv file with the JSON text as a backup. The overall trajectory of errors detected with the default settings is shown in Figure 8.12.

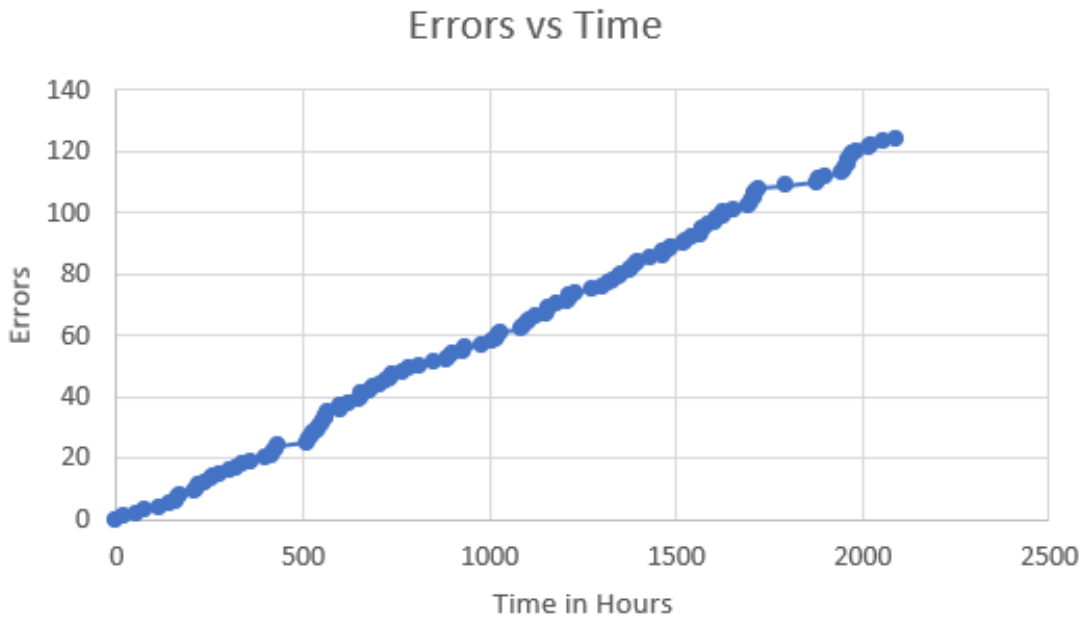


Figure 8.12: Time history of errors detected in the SysML simulation model

Table 8.1: Summary Statistics for Error Detection Exponential Fit

lambda	loc	scale	mean	variance	skew	kurtosis
0.0725	3.0	13.7903	16.7903	190.173	2.0	6.0

By taking the difference between the times at which errors were detected, a set of samples of interarrival time are produced. The histogram showing the frequency of interarrival times is illustrated in Figure 8.13. Source code for processing these interarrival times, as well as for the following data, is given by Appendix A Section A.2.

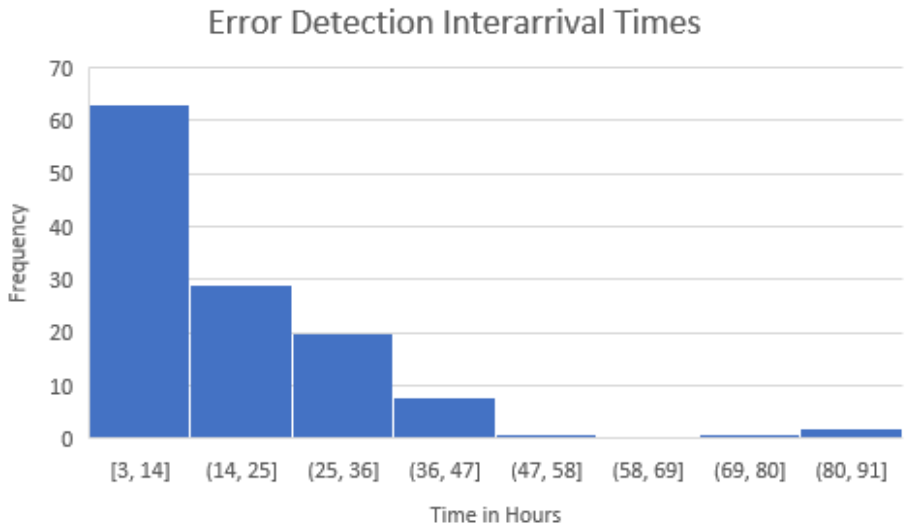


Figure 8.13: Error Detection Interarrival Time distribution from the simulation

With the data on the interarrival times, it is possible to fit a distribution to the errors. In terms of a failure model, the exponential distribution is used here, although it may not capture the long tail effects as well. Some other distribution like a modified Weibull distribution might capture the full behavior more clearly with additional data processing. For the exponential distribution fit, the statistics are shown in Table 8.1 and a comparison of the fitted distribution to simulation data is illustrated by Figure 8.14. Note that there are differences in the histogram binning between Figure 8.13 and the orange Simulation Data boxes on Figure 8.14.

The results shown for the error detection process should not be surprising. For

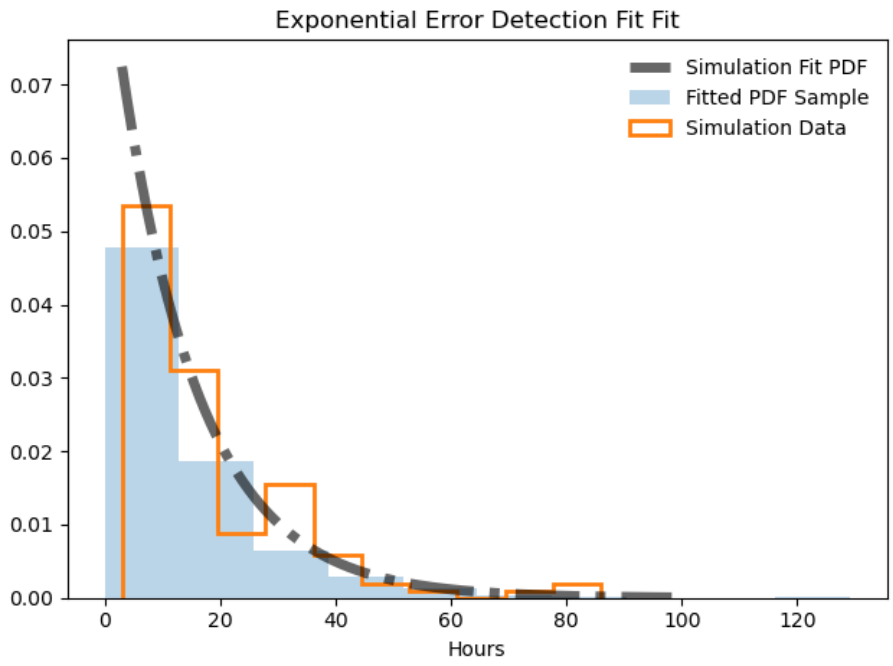


Figure 8.14: Exponential Distribution Fit comparison to Simulation Data

Table 8.2: Summary Statistics for Requirement Revision Exponential Fit

lambda	loc	scale	mean	variance	skew	kurtosis
0.0203	24.0	49.143	73.143	2415.02	2.0	6.0

this kind of event-driven model, the expectation would be that an exponential distribution of some sort might characterize the interarrival times for the phenomenon being modeled. However, important to note is that this simulation is prescribed in the system model, and as such it may be possible to serve as a tool for SE planning purposes and quality control. The finding here is not that this process is characterized in this way, but that in part this representation may serve as a generalized basis for the validation process concerns in project planning and may be represented in the system model as such.

In this sense, consider the requirement revision data plotted in Figure 7.5 from Experiment 1. While there are issues with treating this data as a Poisson process due to simultaneous arrivals – an artifact of the granularity of the measurement — a similar analysis as the error detection can be performed by considering the event

Table 8.3: Summary Statistics for Tuned Error Detection Fit

lambda	loc	scale	mean	variance	skew	kurtosis
0.03704	18.0	26.9918	44.9918	728.5574	2.0	6.0

as the time between any recorded requirements revisions. Figure 8.15 illustrates this revised portrayal of the revision data, and Table 8.2 gives the summary statistics for the requirements revision distribution fit.

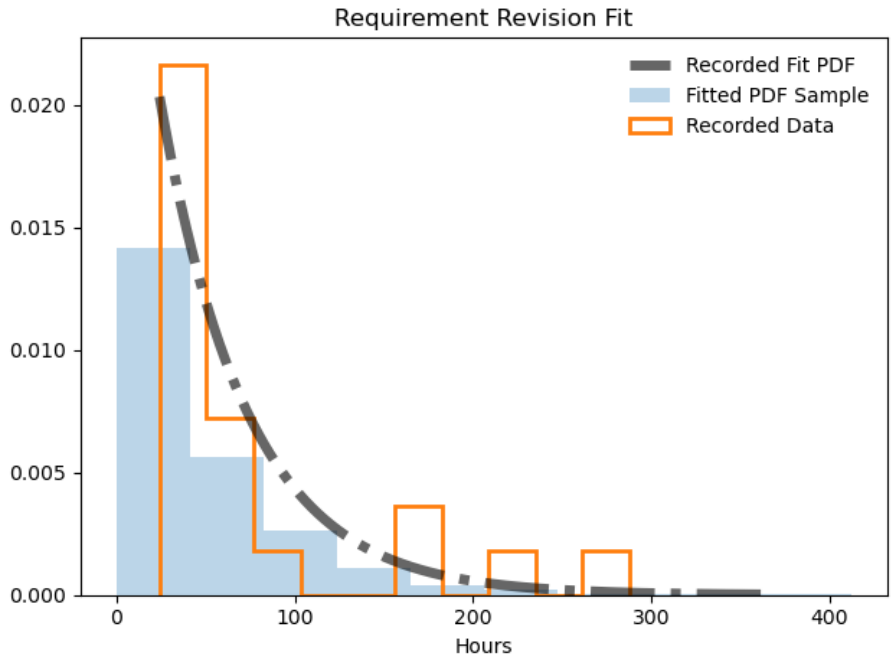


Figure 8.15: Exponential Distribution Fit comparison to Requirements Revision Data

The general shape of these plots is quite similar, and clearly the scale is the main difference. It is possible to try tuning the simulation by changing the decision node probability and the node durations. Figure 8.16 illustrates an example for $p_{error} = 0.3$, $d_{pt} \in [15, 30]$ h, and $d_{eh} \in [15, 30]$ h.

Note however that this attempt at scaling up the simulation couldn't quite match all the statistics, as presented in Table 8.3, compared to Table 8.2. Additionally, the total number of recorded revisions and the overall duration were too low and too high, respectively. A simulation model more representative of the expected process could

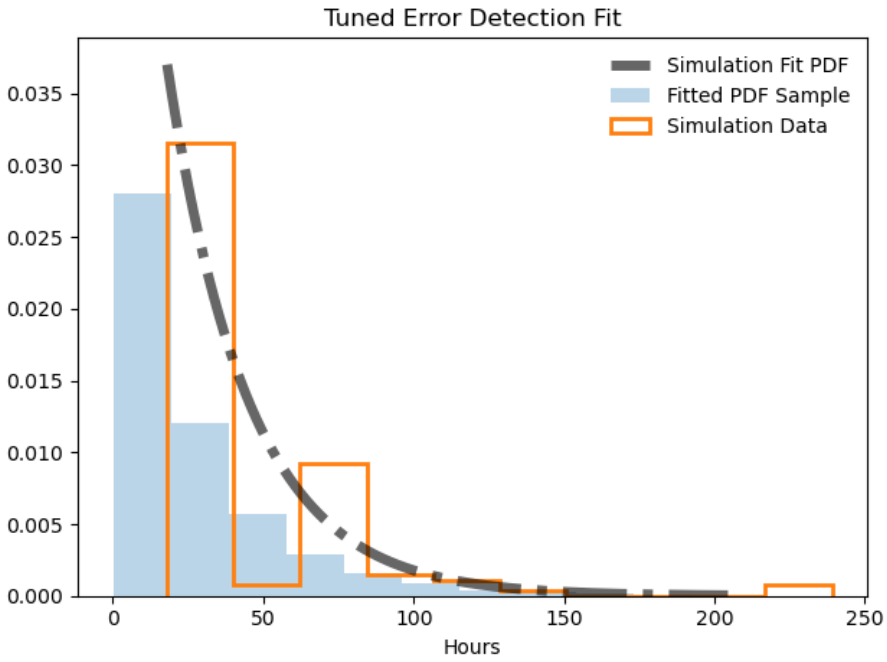


Figure 8.16: Attempt at Tuning Error Detection Simulation

be formulated in future work by changing assumptions or improvement the network, and provide a better match to the processes seen in real requirements revision. Importantly, simulations like these can give the expected number of errors detected in a requirements set and the duration of the effort. However, it is important that the simulations be tuned for the expected conditions, and the status re-evaluated as the process continues.

While these simulation techniques appear promising for potential planning applications, they unfortunately do not provide much insight into task proposals. In order to provide deeper insight into task proposals, the techniques of DES and ABS mentioned in Hypothesis 2 need further definition.

8.3 Simulation Formalisms

Simulation techniques for process models are varied, especially in the domain of planning. DSM, critical path, and other simulations might be proposed. For example,

Blanchard and Fabrycky define the critical path method (CPM) and related program evaluation and review technique (PERT) as a network of tasks where "For each completed network, there is one beginning event and one ending event, with all activities leading to the ending event"[12, p. 673]. Thus CPM and PERT are primarily interested in task networks which are directed acyclic graphs, that is, they have no repetitions or iterative sets of tasks. As defined earlier, many task definitions in the requirements validation domain are iterative in nature and involve cycles. For networks of this nature, as in SA, other analysis capabilities are necessary.

In this case, the interest is towards certain types of business process models which might be composable, in particular SD, DES, and agent-based models[46]. Both agent-based and DES are of interest for process-planning work as they have a granularity which is more directly applicable to tasks. Bott and Mesmer propose leveraging archetypes of engineering cognition in formulating a markov process governing agent behavior[116]. While this could prove a useful component in SE simulation and analysis of the design process, it is less concerned with prescriptive task descriptions and more concerned with the final system design quality. The issue at the core of this thesis is about providing better discrimination amongst prescriptive task proposals. This leaves DES, in which it may be possible to define as events the elements of the prescriptive task proposal. For DES models, it is important to consider the simulation environment[56]. There are several options for DES, ranging from custom to open-source to commercial implementations.

8.3.1 Canonical DES Simulator and Functional Paradigm

One key consideration for simulation is the ability to operate on large sets of data. For example, consider a simulation of SE tasks involving many different steps, or even a simple simulation model run for many different cases. The first source of computational cost is a result of the SE method proposal; while the second is a result of the

need to explore the space defined by the method proposal to predict performance. To counteract these sources of computational cost, consider what was shown by Iacobucci in the power of Map-Reduce to process large amounts of data and tackle the combinatorial explosion[117]. Thus, let the first experimental hypothesis here be that if the DES may be parallelized, then real-time exploration will be possible; and that further, if DES is to be parallelized, then a functional implementation will be best suited for this exploration.

While [56] provide rich detail on the theory behind simulation techniques, they do not provide the most concrete foundations. In contrast, a canonical event-driven simulation suitable for the first experimental hypothesis is available in [118] for a digital circuit model. Many re-implementations of this digital circuit model exist, including adaptations to other languages (the original was in Scheme, and other language texts provide equivalent programs[119]). The version of the digital circuit by [119] is in Scala, which is a language known to be used interoperably with MagicDraw e.g. various projects such as dynamic scripts[120], which is presently about 99% Scala. An additional temptation of this implementation is the potential for a "low-level" integration to MagicDraw, an integration with direct API access which would limit file read-write operations.

The canonical event-driven simulation of the digital circuit demonstrates the elegance of the functional paradigm; however, the canonical implementation is an example of mutable data. For parallel execution, the implementation must be refactored in terms of immutable data. This refactor is made more difficult by the lack of experience of this particular author and constraints in terms of scope of the thesis and duration. While it may be possible, without a direct route to parallel simulation, familiar implementations of DES will be more expedient even if there is some overhead in operation. However, note that implementing the DES such that it has API-level integration, regardless of functional paradigm, holds much promise for future efforts.

However, the method simulations will be constructed in Python using simpy.

For the simulator construction, there are several desirable characteristics to address the method formulation problem:

- Adaptable to multiple SE methodologies as inputs with:
  - Different numbers of tasks
  - Different loops or iterative processes
- Non-proprietary input format
- Easily translated from SysML model
- Accepts multiple parameterizations of processes

These characteristics enhance the utility, portability, and applicability of the simulator implementation.

8.3.2 Basic Simulation of SE Tasks

First, consider the Spacecraft Requirements Definition (OOSEM-lite) from Figure 8.3. Constructing a basic simulation of this process in simpy is relatively straightforward and requires that each task be represented by "yielding" the subsequent task. The setup may be relatively straightforward just to demonstrate the implementation. The code is in the Appendix as Listing A.6, and the output is given in Listing 8.1.

Listing 8.1: Simple SRD Simulation Output

```
creating environment for:
::OOSEM-FireSat System Requirements Definition::
srd created, running sim...
1. Defining external interfaces for mission elements
   ...
```

```
2. Specifying spacecraft behavior supporting
   required functions and states
3. Identifying potential failure modes
4. Specifying spacecraft performance, physical, and
   quality characteristics
5. Capture system requirements in terms of a black
   box specification to refine text-based
   requirements
end sim
```

While Listing 8.1 displays a relative simplicity, note that for the SA Model Development (SA-MD) process from Figure 8.6 that is not always the case. Listing 8.2 gives the output of a single run of the SA-MD tasks. The number of iterations depends on the quantity of state variables discovered during the process.

Listing 8.2: Simple SA Model Development Simulation Output

```
creating environment for:
::SA Model Development::
smd created, running sim...
1. Identify Needs
2. Identify State Variables
3. Define State Models for State Variables
All state variables consumed - modeled
4. Identify Measures for State Estimation
5. Define Measurement Models
6. Identify commands to control state variables
7. Define command models
Step 7 discovered state variables!
```

```
2. Identify State Variables
3. Define State Models for State Variables
All state variables consumed - modeled
4. Identify Measures for State Estimation
5. Define Measurement Models
6. Identify commands to control state variables
7. Define command models
end sim
[('init', 10), ('Step 2', 11), ('Step 3', 0), ('Step
   7', 2), ('Step 2', 3), ('Step 3', 0)]
```

Thus, the actual task description is a sort of random variable, whereby steps in SA-MD may be repeated any number of times as long as new state variables are discovered in Step 7. In the simulation run from Listing 8.2, it so happens that on reach Step 7 the first time that two additional variables were identified, but on the second iteration Step 7 yielded no new variables. As this is a random process in general, a subsequent simulation may have different characteristics, such as Listing 8.3.

Listing 8.3: An additional run of SA Model Developmenet

```
creating environment for:
::SA Model Development::
smd created, running sim...
1. Identify Needs
2. Identify State Variables
3. Define State Models for State Variables
All state variables consumed - modeled
4. Identify Measures for State Estimation
```

```
5. Define Measurement Models
Step 5 discovered state variables!
6. Identify commands to control state variables
7. Define command models
Step 7 discovered state variables!
2. Identify State Variables
3. Define State Models for State Variables
All state variables consumed - modeled
4. Identify Measures for State Estimation
5. Define Measurement Models
6. Identify commands to control state variables
7. Define command models
end sim
[('init', 10), ('Step 2', 11), ('Step 3', 0), ('Step
   5', 3), ('Step 7', 6), ('Step 2', 7), ('Step 3',
   0)]
```

There are shared characteristics between these simulations. For example, consider that a simple desire for the OOSEM SRD simulation might be to capture some duration as a random variable that accumulates across each task. Likewise, the state variables are a random variable which evolves from one task to another of SA-MD. To keep assumptions low, the variability might be taken as a uniform distribution between a minimum and maximum value. Finally, a key issue in the complexity of the solution procedure here is that based on these characteristics about the task simulation, the actual semantics of the simulation program may differ. That is, different program routines might be called at different times. In the worst case complexity, the solution for the simulator would be some form of compiler. However, an interme-

diate level of difficulty will be pursued here to define the semantics for simulating the method proposals.

8.4 Establishing A Domain-Specific Language for Simulation

For each SE methodology, a task is a process defined by a description, a series of actions for the simulator to take, an ID indicating the next process or task, and some data associated with the task. The description for an individual process would be the same as shown above for the basic implementation of simulating the tasks, a short description of what the step entails. The actions are actions which the simulator must perform — a critical piece of infrastructure is a common definition of actions which may be specified. For this purpose, a Python dictionary named `ACTIONS_REGISTER` provides a set of names and corresponding implementations.

8.4.1 Permitted Actions: What Can Be Expressed

One basic action is the `print` action. In the later demonstrations, `print` is used to print the task description as in the previous example result listings. The action itself accepts any text as a single input variable.

Some algebraic actions are provided. The actions `add`, `increment`, and `uniformAddInt` provide simple math capabilities. The action `add` takes two inputs and adds their values, while `increment` adds 1 to a single input variable. The `uniformAddInt` action takes as its argument a list $\bar{x} :=< x_1, x_2, x_3 >$ such that x_1 is an initial integer value, and a uniform random integer selected from within the bounds $[x_2, x_3]$ is added to the initial integer value.

Three actions have complex meaning which depends on simpy. The `success` action applies the simpy `success` method to the input which must be a simpy event or process type. The `yieldtime` action takes an amount of time and the current simpy environment, and yields a simpy timeout event of the input duration. It is usually

necessary to conclude a simulation specification in this simulator implementation by the `yieldtime` action with a trivial value of 0, in order to enable the simulation library to generate a sort of null event that concludes the simulation. Finally, `nextproc` uses a task definition function, the simpy environment, and a passthrough value to yield the next step in the simulation.

Finally, an action with no implementation named `condition` is listed in the `ACTIONS_REGISTER`. The `condition` action modifies the behavior of the parser to obtain, evaluate, and follow-through on the results of a conditional statement provided as an action to the process node/task. In the specification file, a `condition` is defined using a test, a block to run if true, and a block to run if false. These blocks must leverage the actions defined above. However, the functionality of this action is highly coupled with the parser implementation.

8.4.2 Parsing Capabilities

The software architecture for the simulator program is relatively straightforward. After loading a dictionary from JSON, the dictionary is used to establish a list of ProcessNodes in a node chain (`GeneralDES().make_nodes()`). These nodes are sorted according to ID, and then the manager class sets the previous and next node properties for each ProcessNode. Actions and Data are stored within the ProcessNodes for later usage. Importantly, a simpy environment is passed along at all points in this process so that the manager class and ProcessNodes each are aware of the same environment. After the preliminary parsing of the JSON and creation of the node chain data structures, the primary parsing activity begins by requesting that the simpy environment processes the first node.

The primary parsing capability is used in establishing the tasks performed by the ProcessNode when the node is called as a function. As part of the structure of the generators involved, the actions of the node will be parsed according to various

conditions on the action name and/or action body. An action body of "descr" will make the node description available for printing. Integer action bodies occur for `yieldtime` actions. The `nextproc` action is the most crucial, and it leverages a helper function to generate a subsequent node, but this is done in two pieces according to whether the action body is "nextid" or some other node, as is occasionally the case for SA-MD. The capability of `nextproc` should be to step to any node in the node chain by ID, but it has only been tested in the context of SA-MD with small changes in the kinds of nodes targeted. Fields from the node data are made available if the action body is "passval,minval,maxval". The `condition` action is the most complicated, and is parsed first according to the "test", which is handled effectively as $(control, (comparator, value))$. With *control*, the item from the node data is accessed for the comparison. The *comparator* must be compatible with the Python operator fields. In the case of SA-MD, this is "gt" for "greater than". Finally, *value* is the quantity of interest, so 0 for SA-MD where the condition is $SV > 0$ for iteration to step 2. A true result for the "test" body runs the "iftrue" body, while false runs the "iffalse" body. Either case returns a tuple of appropriate actions and a generator to build the subsequent steps for simpy, which can be parsed into any of the previous actions.

Of course, for the most part these parsing steps are handling *arguments*. The parsed arguments are used by the node while generating a task:
`ProcessNode().generate_tasks()` to "act_on_it". Specifically the action name and the parsed body are used according to the `ACTIONS_REGISTER` for the simulation to do something.

After all tasks have been generated and registered with the simpy environment, the parsing capability ends - unless called upon again during simulation due to potential alternatives.

8.4.3 Configuration

The simulation itself is configured via a simpy environment. The random number generator seed is not fixed. As the capabilities previously described are provided as a command line tool, a wrapper is necessary to drive automated runs of the simulation. This is done using a `CommandWrapper` which launches the simulation file with arguments as a system process. The wrapper returns a generator the size of the desired number of runs, each time returning a system process. In this way, 1000 or 10000 runs of the simulation can be queued for later analysis.

The simulation runs are configured with JSON files. For OOSEM-lite SRD, each process has a "duration" passthrough value which is a random uniform integer between 5 and 10. In this case, it is more that the passthrough value is considered as a duration, than that a duration is explicitly modeled in the simulation. For SA-MD, passthrough value is used to indicate the number of state variables passed from one step to the next. Unlike SRD, SA-MD can be iterative in nature. Thus, the history of state variables and the number of iterations are two measurements which can be taken for SA-MD. Importantly, SA-MD and SRD are partly responsible for the same work as discussed previously, since SRD is applying and using models in the specification of new spacecraft requirements. For SA-MD, the tasks begin with 10 state variables. The "Identify State Variables" step adds $[0, 2]$ as a uniform random integer. The "Define State Models for State Variables" step removes state variables as a random uniform integer in $[-10, -2]$. The "Define Measurement Models" step adds back a random uniform integer in $[0, 4]$. Finally, if the "Define Command Models" step receives a positive integer of state variables, it adds a random uniform integer in $[0, 4]$ before stepping back to "Identify State Variables".

All source code, including the full input file definitions, is given in Appendix A Section A.2. This includes simplistic DES of the methods, the generalized DES parsing tool/simulator, as well as tools to automate and analyze data.

Table 8.4: OOSEM-Lite and SA data from Repeated Simulation

	OOSEM-Lite SRD	SA-MD
Number of Samples	1000	1000
Mean Duration	34.93	n/a
Duration Std Dev	3.0589	n/a
Mean Iterations	n/a	4.899
Iterations Std Dev	n/a	2.8696

8.5 Simulation Results

After running the simulations 1000 times each, several statistics were calculated according to the data available. Due to the limitations of the implementation with respect to the passthrough quantity, a quantity passed from one process to the next, there are comparability issues between the data. This discrepancy can be seen in the summary available in Table 8.4. The same items are not measured here from one method simulation to the next. However, despite the limitation of the implementation, consider that by assigning an average duration to an iteration, one might obtain some similar results. If an organization knows the typical model development duration, then statistics on time can be obtained in terms of iterations. For the data generated here however, the comparisons are more limited. A box plot characterizing the duration of the SRD process is given by Figure 8.17. Similarly, a box plot characterizing the SA-MD iterations is given in Figure 8.18. Figure 8.19 provides a box plot on the SA state variable history. Note, it is possible to obtain both quantities in the case of state analysis via post-processing of the simulation output text. The passthrough variable in SA is the number of state variables from one step to the next. However, iterations are measured by counting the number of times the process repeats step 2. In SRD, the same steps are always completed, and there is no distinguishing element run to run in the output except for the duration quantity. Figure 8.20 provides the distribution of histories or trajectories of the number of state variables, which are updated in specific process steps of the SA-MD process. By performing

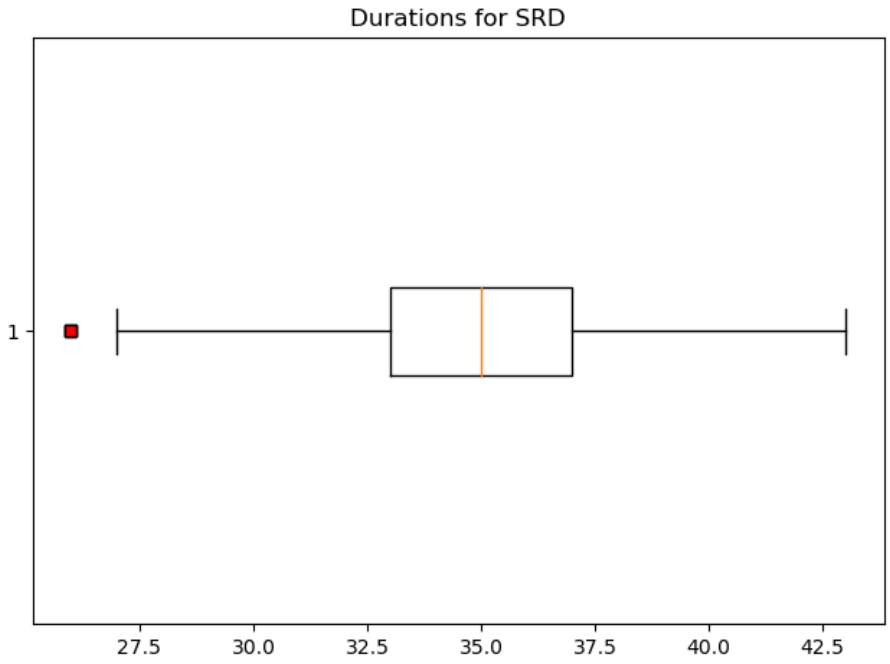


Figure 8.17: OOSEM-lite SRD Duration Box Plot

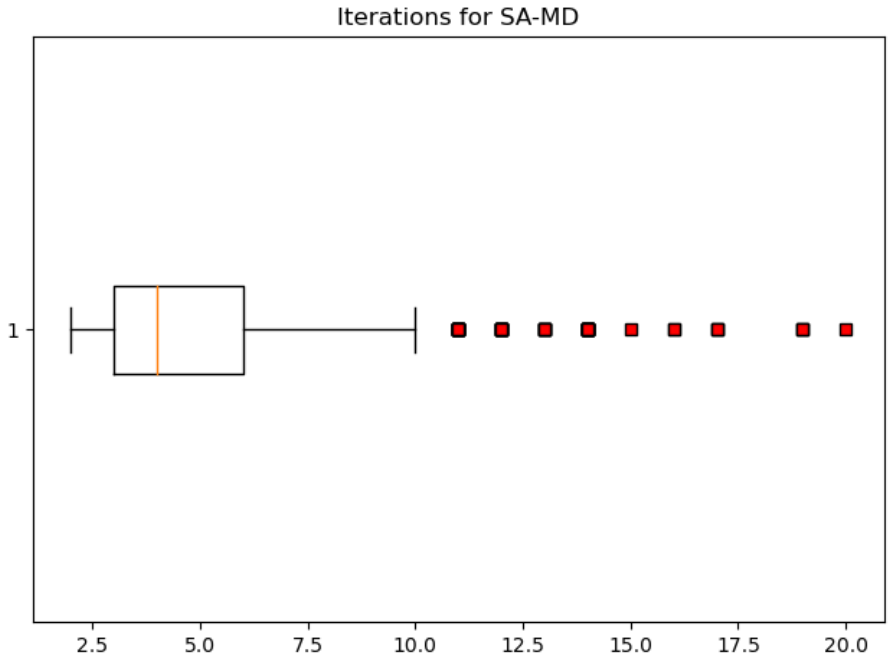


Figure 8.18: SA MD Iterations Box Plot

a sufficient number of these simulations and obtaining sufficient calibration data, a planning team might be able to establish a probability of completion within a win-

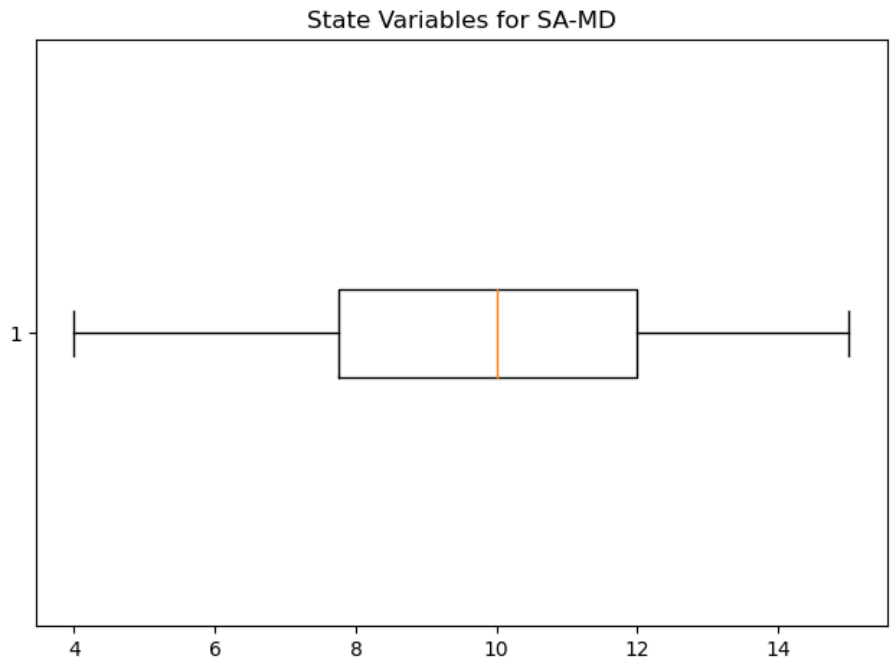


Figure 8.19: SA MD State Variable Maximums Box Plot

dow of time, e.g. 3 iterations, 4 months, etc depending the kind of data available, as well as whether such data is reasonably measured according to the initial number of relevant state variables. Important also in calibrating these simulations is the effect of the initial conditions in the specification files. In the statistics for SRD, the mean duration is very nearly 35 at 34.93, or $5 * 7$, five steps taking 7 units of time, where each step is a uniform integer in $[5, 10]$. Therefore this result matches the expectations set by the process specification file. The iterations quantity for SA-MD is less obvious, as the process ends only when the number of state variables goes to zero. While in each iteration the number goes to zero temporarily, settings on the probability of new variables being introduced drives the likelihood of additional iterations. One interesting coincidence exists in the resulting data. That is, by choosing an iteration duration of 7 units of time, then the average duration of SA-MD as modeled is approximately 34.293 units of time, just shy of the average duration of SRD at 34.93 units of time. For these simulation runs, the result is purely coincidental and can say

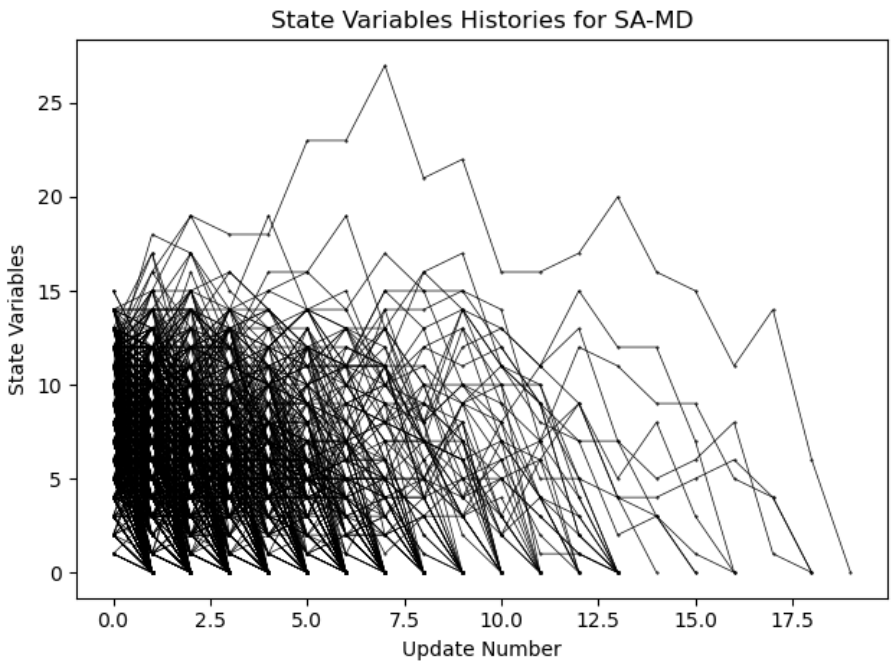


Figure 8.20: SA MD State Variables trajectories over all simulation runs

nothing about the expense in time of either SE methodology; however, it validates the concept expressed above that by properly calibrating the simulation, comparable results may be possible. This calibration of course would require proprietary data on organizational model development activities, and a desire to implement one of these SE methodologies.

8.6 Integration Possibilities

Finally in the proposal for this task, consideration is to be given to system model integration. The format of the general DES simulation input specification is a json file. The output format is text received on the standard out and standard error pipes. This text can be gathered and processed to obtain the data presented in the simulation results. Creating the JSON file is relatively straightforward but will depend on the particular elements chosen in the system modeling tool. Parsing the output text stream within the system model will require similar code as has been used

here to exist within the system modeling tool. In all, it is very possible to integrate this simulation. However, the important aspect of the simulation is that the meaning of the output varies based on the input context - not all simulations will be measuring duration. This integration, as well as potential integration of Experiment 1, is to be investigated further in Experiment 3b.

8.7 Summary and Conclusion

Experiment 2 involved a particularly challenging software implementation. It is likely that an approach aiming at creating a compiler would be more flexible in simulation. The approach chosen is a compromise between the objective of a DES flexible in terms of SE methods, yet implementable in the time, skill, and scope of the thesis. As the simulation does not depend too much on the process description — other than in placing limits on mathematics such as uniformly distributed integers — many processes may be modeled and simulated by this tool. In the context here, only SE methodologies are of interest and the OOSEM-Lite process from [98] on Spacecraft Requirements Derivation is presented alongside SA-MD.

Overall, Hypothesis 2 is accepted. SE methods are clearly amenable to simulation by DES, as they are strongly driven by event-focused terminology. Further, as discussed in Experiment 1, SE methods and in particular requirements validation/requirements status is industrially dependent on event-specific markers. Unfortunately, these facts lead to the rejection of Hypothesis 3; the simulations for SRD and MD do not provide any quantification of error detection and therefore cannot be substituted within the SD model of Experiment 1. At best, these simulations may be replacements, as modeling a path from an initial state of work to be done to work completed, especially in the case of a specification like for SA-MD. In particular, the use of iterations for SA-MD and time (units unspecified) for OOSEM-Lite are interesting for task planning as they can be proxies for cost estimation in terms of person-dollars per

hour or person-dollars per iteration. Therefore, regarding Hypothesis 3, the bi-level simulation is what is rejected; utility is still expected for project planning, especially upon integration with a system modeling environment.

9

EXPERIMENT 3: INTEGRATION OF MODELS

9.1 Recap of the Experiment

The third experiment has two parts. After formulating a small number of method models in Experiment 2 and testing them for their behavior independently, the method models must be interfaced to the System Dynamics model from Experiment 1. Experiment 3a corresponds to Hypothesis 3. If Experiment 3a succeeds, then the hybrid model will provide requirements volatility trends as a function of SE tasks. However, should it prove impossible to combine the method simulation model with the SD model in a hybrid simulation, these models are still useful individually for the overall purposes of this investigation as discussed above. A hybrid simulation model may provide better qualitative benefits to decision-makers by giving different levels of abstraction for business operations, and if this experiment succeeds, such a model will be produced. However, if the experiment fails and the method models are instead an alternative to the System Dynamics model, then they can be compared. Overall, the objective is to investigate Hypothesis 3:

Hypothesis 3 If SE methods are treated like other business processes, then bi-level hybrid simulation will provide better parameterization for task-planning purposes.

In the first portion of this experiment, the proposed procedure was to:

- If the models built thus far are compatible, construct an appropriate software interface for co-simulation

- If the models are incompatible, provide a comparison of the calculation of requirements volatility in terms of validation tasks vs. the Walworth et al characterization of the organization as a learning system.

The second part of Experiment 3 relates to representation. That is, how to manage the variation of the individual and/or combined models. While some variation might be based solely on parameters, DES models of tasks have a high combinatorial degree of freedom when considering possible modifications of the directed graph. Additionally, as with Sprock and McGinnis 2015[65], it is desirable for the planning of the SE tasks to be tightly integrated with the SE model or documentation (as discussed by Martin for the document-based case[4]). To enable these capabilities, the simulation environment should be created according to a system model description, per Experiment 3b. The choice of SE language is only important insofar as the tools and environment of PMTE are convenient, as is the case of SysML for this author. While the specific transformation will be different in implementation from other languages, or possibly even the same language in different tools and environments, this author has developed several capabilities in an existing set of tools which may prove useful for building Experiment 3b. The objective here is to provide a direct capability for systems engineers to leverage the models developed across Experiment 1 and Experiment 2. The proposed procedure included:

- Develop a system model
- Create simple simulations of the system model, perhaps using a tool such as Cameo Simulation Toolkit
- Specify stricter transformation rules to the simulation tools used in previous experiments
- Extract simulation models according to transformation rules

These portions are divided into parts A and B. Experiment 3a will be investigated first.

9.2 Experiment 3a: Investigation of hybrid simulation for DES and SD models and comparison of outputs

Previously in Experiment 1 and Experiment 2 it was determined that the DES and SD models were not compatible for including the results of the DES model as part of the SD model. Real data holds up the portrayal of requirements development activities as being event driven along the lines of Jackson et al[70]. With the end result of using a discovery factor on the SD model for the rate equation from undiscovered rework to known rework, the best potential match in terms of error discovery is excluded, especially due to the need for the result of the error discovery model to be integrated during solving of the differential equations of the SD model. This precludes the possibility of tight integration of the various mathematical models as they are currently formulated; however, different formulation and different simulator infrastructure may provide a better means for a combined, multi-level analysis. In this sense, the bi-level hybrid simulation aspect of Hypothesis 3 has already been demonstrated to be incorrect, especially the dubious parameterization of requirements status per the SD model.

There may be some criteria for an acceptable output space for model interoperability. One desirable criteria is for the integrated model to be compatible with numerical integration, so that it would take some number of tasks (e.g. undiscovered rework) and the current time and calculate a rate or number of tasks available at the current time. This might be achieved using a model of a distribution such as an exponential distribution, but the key thing is that the output should be a function of at least those two particular inputs for the Walworth model, ignoring the other parameters for the time being. This hypothetical model would also need to be com-

patible for use in the modeling environment, whether in Vensim PLE or in Python. Finally, it may be desirable to have multiple such models, and provide some means to toggle between them instead of the discovery factor input. This too might be achieved. However, in the current setup for Experiment 1 vs. Experiment 2, the DES models are not strictly calculating an error rate for SRD and SAMD. Instead, the SRD model calculates a duration elapsed and the SAMD model mainly gives a number of iterations, as well as a number of state variables vs some rough alignment of iteration. While the error detection process does specifically find errors, it lacks the compatibility criteria stated above for the software implementation, and it also does not directly map undiscovered rework to discovered rework — a total number of tasks and a set of probabilities might give a number of errors found. So for the current set of event-driven models in Experiment 2, the best alignment is from work to be done to work really done, though this is not exact as the measure is not necessarily always in terms of tasks, depending on the model.

Finally, proposed in this experiment is a comparison of the models as substitutes. In this sense, consider the fundamental hypothesis of the SD model for requirements status on the basis of the S-Curve parameter trajectory. However, according to the SELI Guide [41], not all parameter trajectories are described by S-Curves. Some parameter trajectories may be better represented as a linear "burn-down". The real requirements modification data showed much more linear characteristics in terms of revisions vs. time, while showing an exponential distribution of the duration between modifications. The hypothesis of Walworth et al[42] is not borne out in the data. As such, models like Walworth et al should be used with caution in creating planned parameter profiles and in monitoring progress towards program objectives. Creating a burn-down curve from an event-driven model such as DES may be at least as, if not more, valid, as it may carry a more direct representation of tasks for scheduling purposes with fewer arbitrary parameters about team capabilities which must be tuned

for particular cases, and which cannot be objectively measured. Harder measures should be preferable for modeling, such as time or number of state variables, as these quantities can be defined in such a way that they are easy to measure compared to intensity or learning potential. The focus on measurement is of course oriented towards the practical application of such modeling in assisting the creation of a feedback loop, enabling planning as well as monitoring whether work is proceeding according to plan.

9.3 Experiment 3b: Generate the method models and/or combined simulation from SysML

The models from Experiment 1 and Experiment 2 have very different characteristics. The SD model represents an initial boundary-value problem for the stocks and flows, for which the definite integral is performed numerically for a given interval. The input includes initial conditions and constants, and the output are the final values of the variables in the initial conditions (usually termed state variables here, not to be confused with SA). In this model, the trajectory of the values is returned over the interval, such that four time histories corresponding to the four primary stocks from the Walworth et al model are returned. For DES, the input is a specification file in JSON format with characteristics describing the nodes in the process to be simulated. After simulation, text output characterizing the simulation result is returned to the user. Furthermore, unlike with SD, the DES setup is not one model; the DES is in fact a simulator where the input represents a model. This is an important characteristic. The input to the DES tool is a model of the task proposal, and therefore is directly applicable to the core focus of this thesis. However, note that in Peak and Lane 2014[32], COSYSMO[25] is integrated to a SysML model so that SE planning could notionally be performed from the system model. This entails an additional refinement on Research Question 3, whereby rather than how to represent the method

for simulation, the big-picture issue of how to represent the method as part of a plan in an MBSE language in such a way that the simulator is usable must be addressed. The missing capability is stated as Gap 6:

Gap 6 There is a need to represent new SE process analyses in the MBSE ecosystem, including ones involving method models.

Addressing this integration in Gap 6 issue brings the representation to the domain of MBSE practice, going beyond the model development seen by Walworth, Grenn, and others. To this purpose, Research Question 3a is derived:

Research Question 3a How can the analysis of SE method models be represented in a system model, such that SE planning activities are augmented by analysis in the MBSE environment?

And to the end of answering Research Question 3a: if the external SE analysis environment is rendered compatible on its interfaces with one or more tree-like data structures expressed in tree-like languages such as XML or JSON/YAML (i.e. without reference keys, xpath, etc), then any analysis can be integrated with the MBSE ecosystem on the basis of transforming a template of the analysis data into JSON. First of all, the external SE analysis is for example the software products of Experiments 1 or 2. Secondly, by rendered compatible, that may mean some activity to either 1) wrap a black box external tool or 2) adjust the interface behavior directly (possible in this case due to the nature of the development of Experiment 1 and 2 codes) such that the input and output have some regular syntactical structure. This structure is described as *tree-like*, as for a SysML model of blocks with directed associations only (to any type or datatype), it is possible to use a recursive function which constructs an associative array of key-value pairs or a list of such directly on the basis of specific filters regarding the UML meta-model representation of *owned*

attributes. Such a solution will be described later in this chapter and subsequently reused in slightly modified forms, depending on whether the outcome is a list or an associative array.

The question at hand is now to incorporate the models of Experiment 1 and Experiment 2 in SysML. That is, to be able to run cases for the python codes presented in the culminations of those chapters based on data within the SysML model, and thereby answering Research Question 3a. Due to the different characteristics of these models, different approaches for the SysML integration or interface are necessary, and these differences will be managed by repeated use of the analysis template.

9.3.1 Integration of the SD Model to SysML

The analysis template presented in Contribution [62, Paper B] is applied to represent the Walworth model. Figure 9.1 illustrates the overall organization here. Some

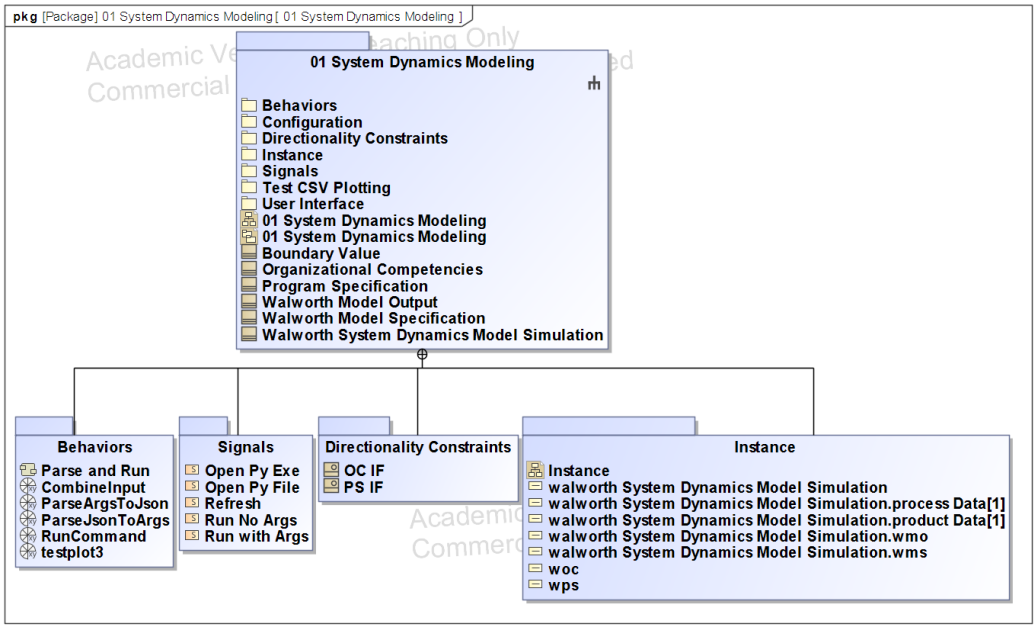


Figure 9.1: Package organization for the Walworth et al model integration to a system model.

items have been added to the template to be used within this setup. Specifically, the template has been rearranged to account for 1) the topic of spacecraft from Friedenthal

and Oster[98] and 2) the idea of "product" and "process" design — frequently termed Integrated Product/Process Design or IPPD as in Blanchard and Fabrycky [12]. In this sense, the SEMP is an artifact of the process. The product constructs specified by Contribution [62, Paper B] are grouped under product data, and additionally the Spacecraft Element is included. This updated breakdown is shown in Figure 9.2; additional stereotypes and customization provide the facility to rapidly make use of these elements in the model.

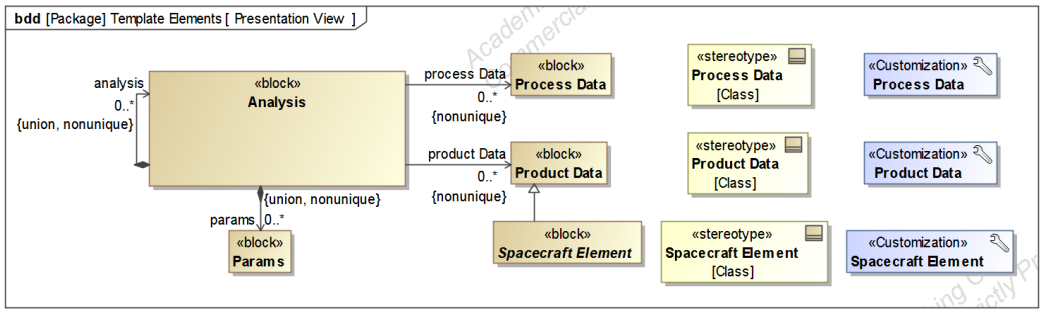


Figure 9.2: Modifications made to the Analysis Template from Contribution [62, Paper B], including product and process elements

Specifying the Process Data

Process data for the Walworth et al model has been divided up between organizational competencies and a program specification, as seen in Figure 9.3. This division has no real importance, other than perhaps in using the model for many different programs under an organization. That is, the values such as intensity, learning power, effort, quality of work done, urgency, attention span, and the discovery factor may not be specific to any particular development effort. Perhaps, if quantification were possible, an organization might be able to establish these numbers for their entire workforce. However, the specific staff allocated to a development effort, the size of the effort, and the start and end points of the effort are specific to it, and not the organization as a whole. Thus, the values are divided among these two process data elements to illustrate the potential different forms of reuse in the system model. Additionally,

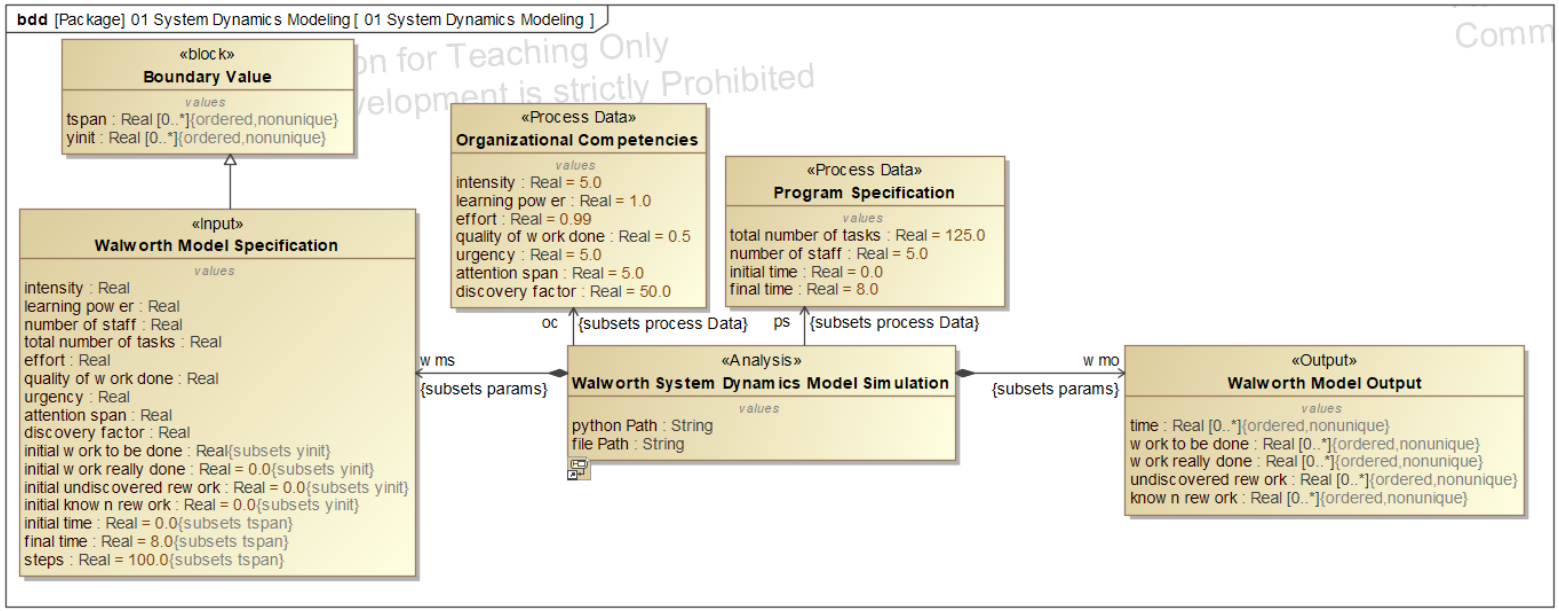


Figure 9.3: Layout of the system model representation of the Walworth et al analysis with divided process elements.

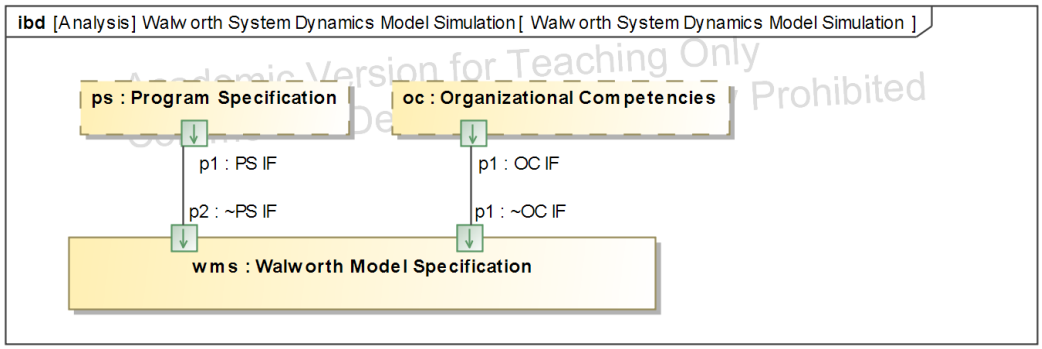


Figure 9.4: Internal block diagram illustrating port connections for one-way data transfer

these values are not to be modified by the analysis. Proxy ports are established with interface blocks which guarantee unidirectional flow of the information during simulation by Cameo Simulation Toolkit. The flow is achieved according to an internal block diagram in Figure 9.4 and a parametric diagram in Figure 9.5.

Analysis Inputs and Outputs

The analysis parameters are defined according to the inputs and outputs. The inputs include copies of all the process data quantities. The outputs include all the time series values from the ODE solver. As time series, these arrays or vectors are an ordered sequence which may have repeated values, or therefore have non-unique elements. Additionally, there is the definition of the boundary values in Figure 9.3, specifically `yinit` and `tspan` which are the arrays of initial states and the vector of time steps, respectively. One quantity which is only found in the input list is the steps, which is part of `tspan`. By giving the appropriate order to these values, the Python code will receive the list it requires to establish the `linspace` of time steps for the ODE solver. Finally, some paths are included at the analysis level, which define the Python executable to be used as well as the Python file to be run.

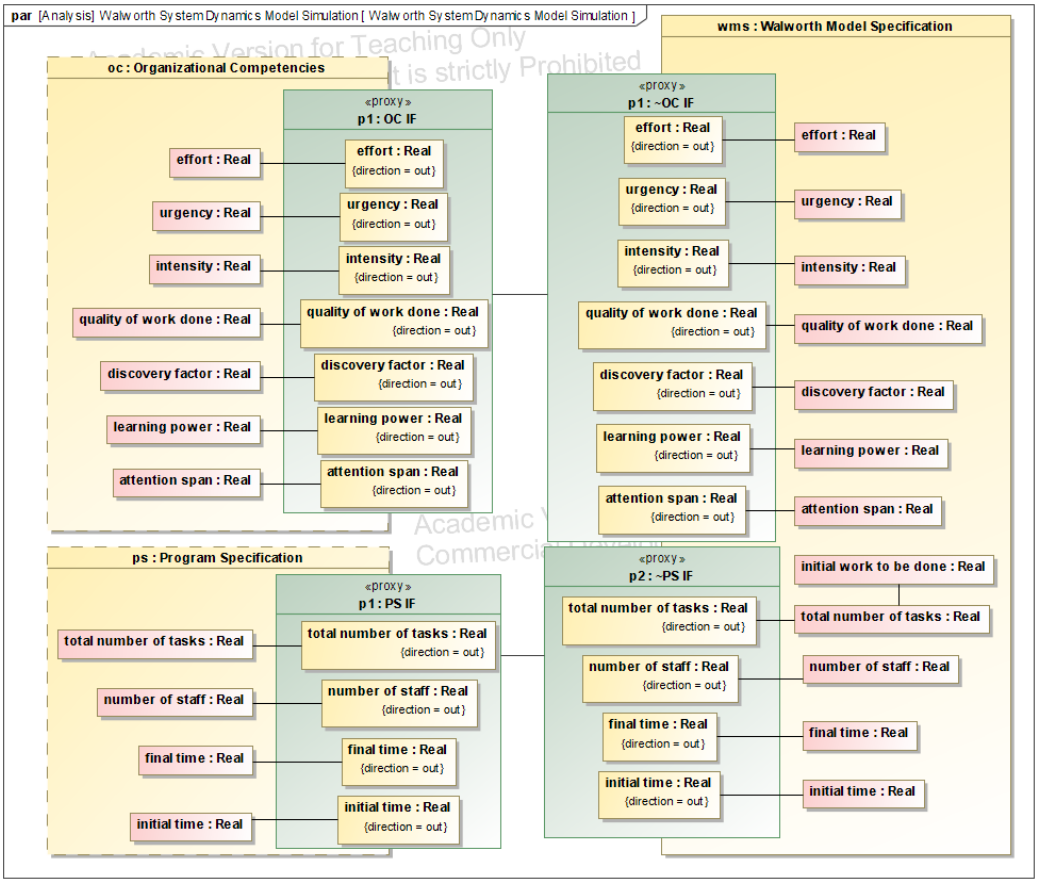


Figure 9.5: Parametric diagram illustrating flow property linkages to values between process data and input quantities for one-way data transfer

Modifications to the Experiment 1 Code

Experiment 1 ran without command line arguments. For Experiment 3, a third revision of the code was created which establishes appropriate command line arguments which may override default values. Additionally, the simulation result is written as JSON to `stdout` directly. The command line arguments will be supplied by the system model — or by any other program — and then the output will be sent as a response. This output can be easily parsed in JSON form and read back into the system model. The modified code is included in Appendix A Section A.1.

Analysis Execution

Several features are necessary for the system model execution. The overall analysis execution is managed by a state machine behavior, shown in Figure 9.6. Notable transitions include selecting various files, as well as running the primary analysis workflow. The analysis workflow involves parsing the system model to the input format, running the system command, and then parsing the result into the system model. The signal to "run with args" will launch the primary analysis workflow. The

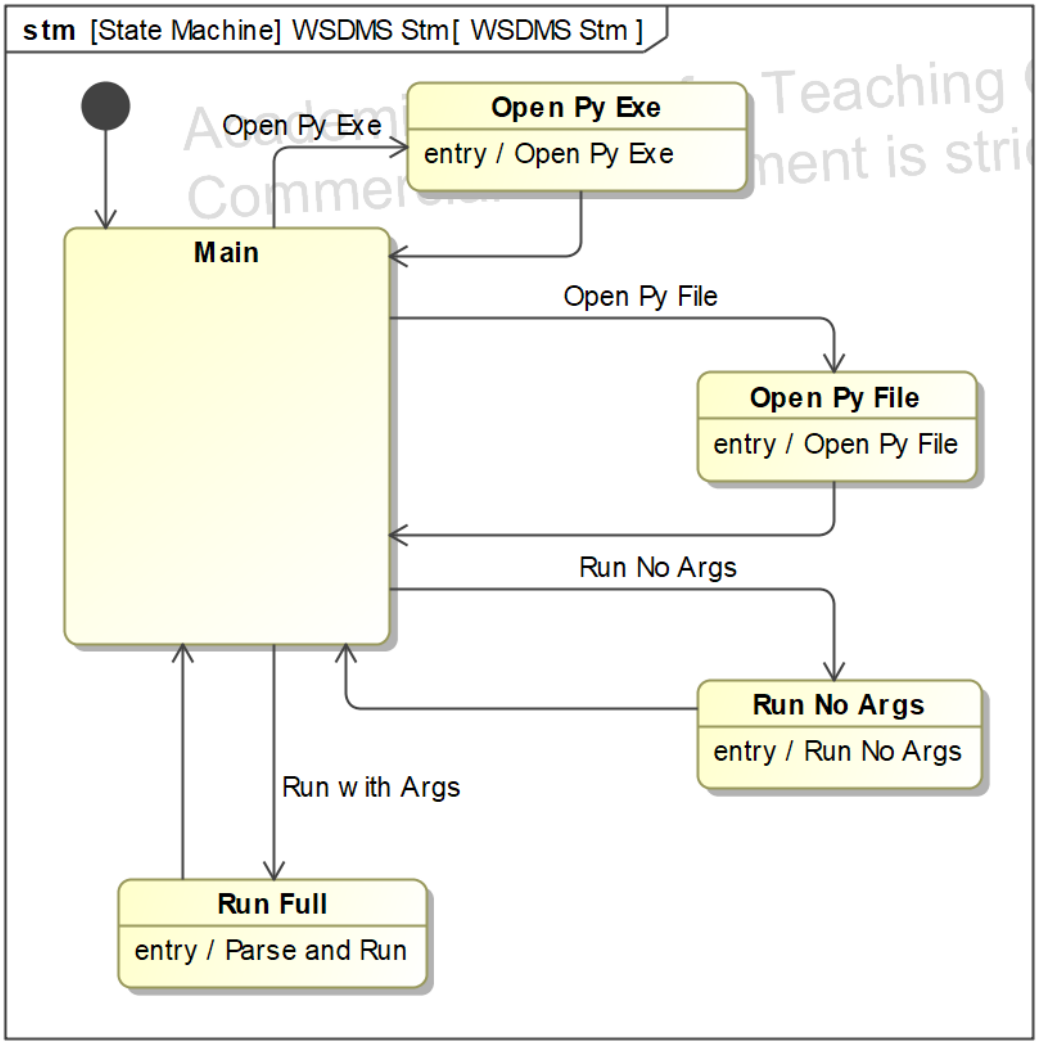


Figure 9.6: State machine governing the behavior of the analysis integration

workflow on the entry behavior is the activity diagram shown by Figure 9.7.

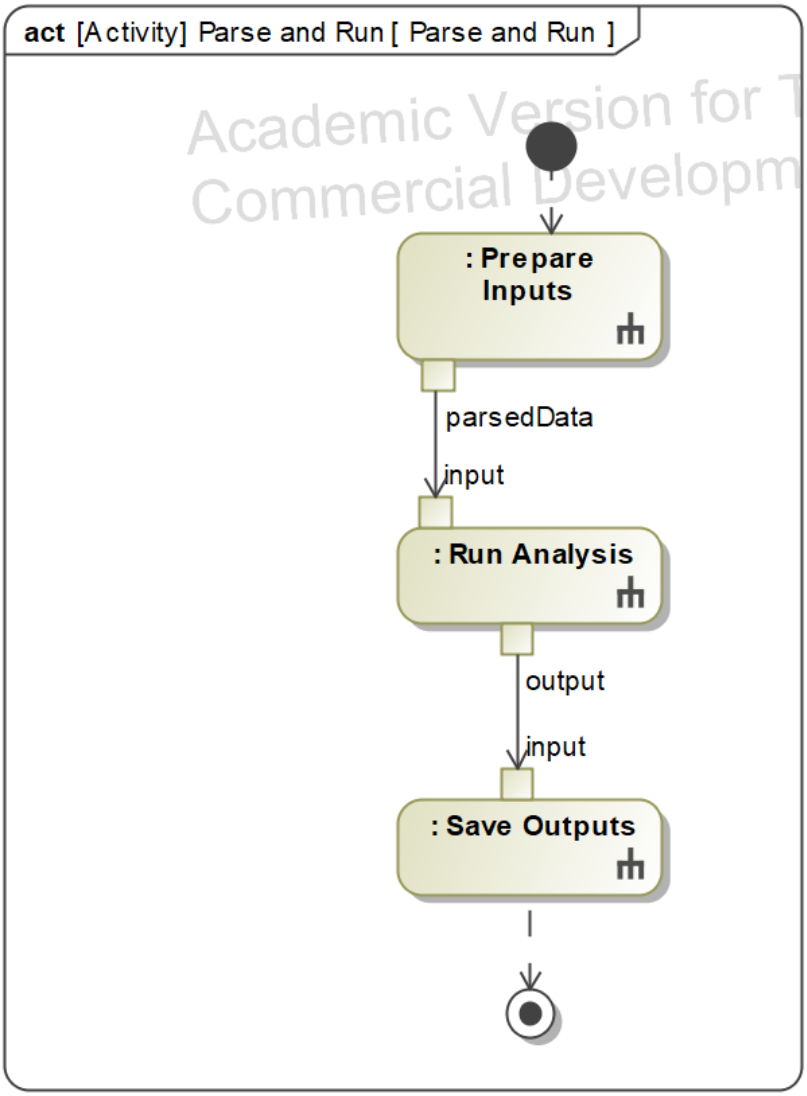


Figure 9.7: Parametric diagram illustrating flow property linkages to values between process data and input quantities for one-way data transfer

Input parsing is managed by the prepare inputs behavior. The prepare inputs behavior is an activity which uses lower-level constructs (fUML and scripts) to extract information from the system model, and format this data to JSON. The activity is illustrated by Figure 9.8. Figure 9.8 relies on scripts to parse the system model. The scripts leverage multiple APIs to scrape data from the model, and then convert that data to JSON. The approach is to first convert the data from the model into an associative array. The scripts perform this action based on the so-called "owned

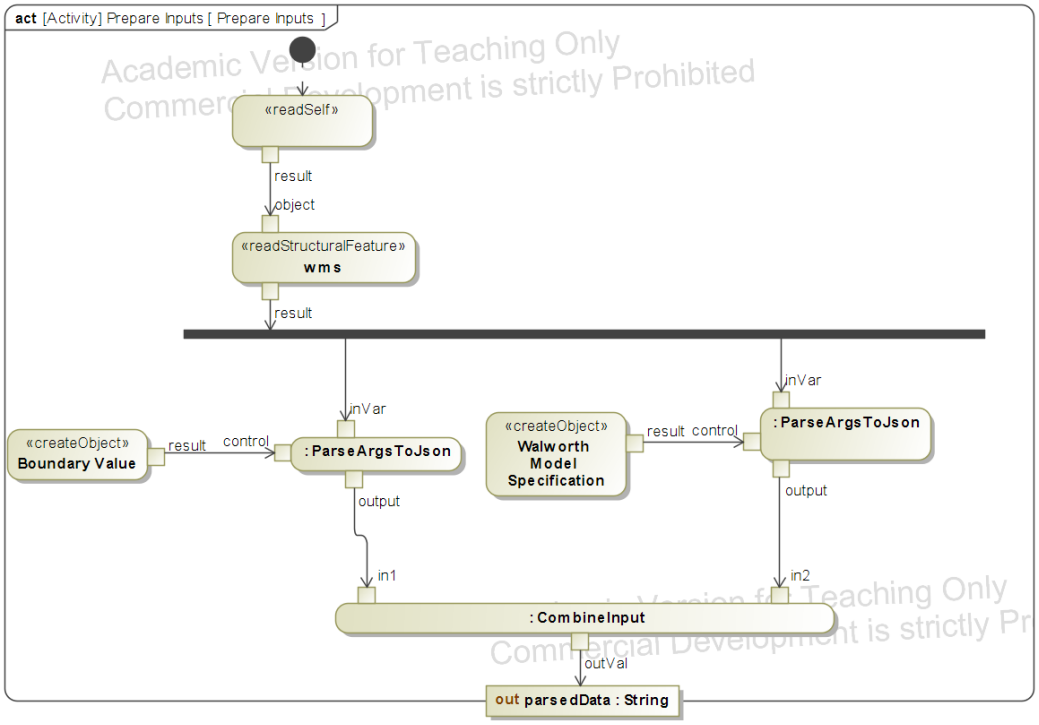


Figure 9.8: Input parsing behavior

attributes" of the blocks in the diagram seen in Figure 9.3. For this reason, the parsing is done twice, to capture both the direct input owned attributes as well as the inherited boundary condition attributes. The associative array is then directly converted into JSON. To piece the two JSON strings together, an additional script reads in the two strings and combines them in a fit-for-purpose way. This new JSON result is loaded into the external Python code over the command line as a Python dictionary type.

The command is handled by another script within the run command behavior, shown in Figure 9.9. This activity takes in the parsed result of the input, the path to the Python executable, and the path to the Python script file and runs a system command. The result is obtained and passed along for parsing.

Finally, as shown in Figure 9.10, the text returned from the system command is parsed. The script used here is slightly more flexible than the script in the input parsing and is able to perform the inverse actions; going from JSON to associative

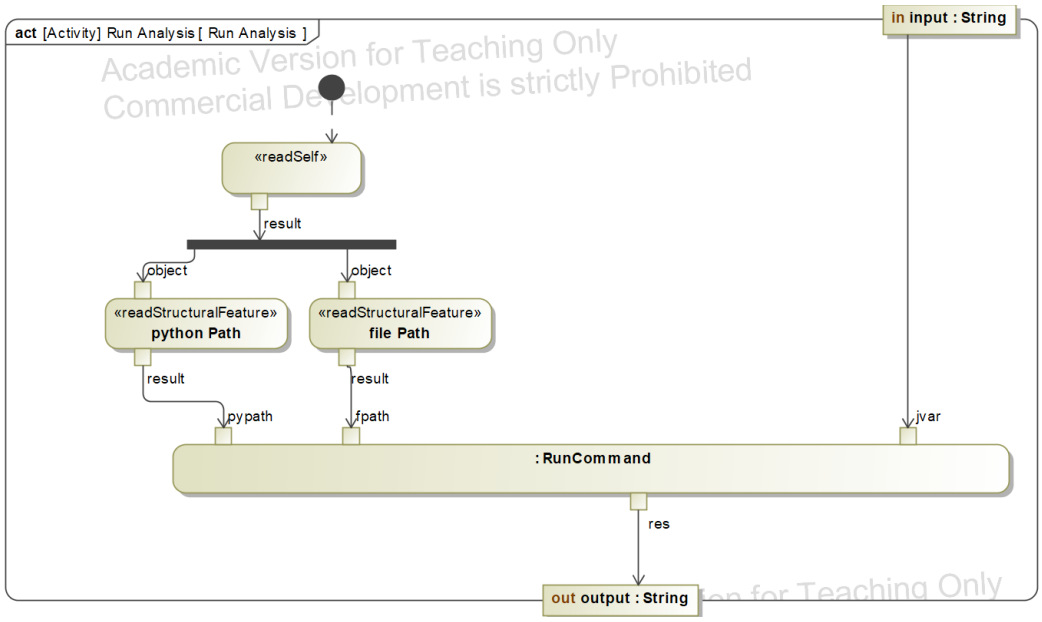


Figure 9.9: Run command behavior

array to the system model. In the end, the time series data is populated into the "run-time" values for the output specification. After the workflow has completed, the

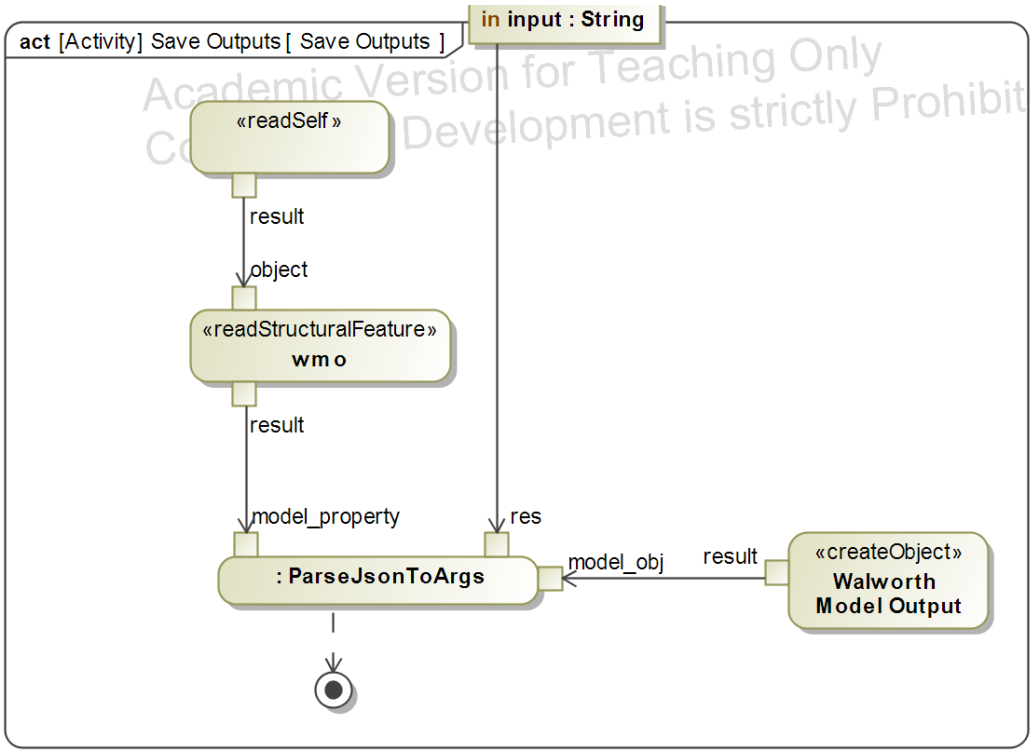


Figure 9.10: Output parsing behavior

state machine in Figure 9.6 revisits Main, and the system model user can choose to save the specific output data of interest. Later, in Experiment 4, these steps can be managed via a graphical interface. However, an important aspect of Experiment 3b is to specify the means by which the system model is transformed into the analysis model content.

In the system model, the inputs are defined in two sets, $BC.OwnedAttributes$ and $WMS.OwnedAttributes$. The term `OwnedAttributes` is part of the system modeling language, and denotes properties which are contained by a block but not inherited by the block. The union of these two sets is the full input set. As far as transformations go, the approach here is centered on the pragmatic. Figure 9.11 illustrates the general concepts.

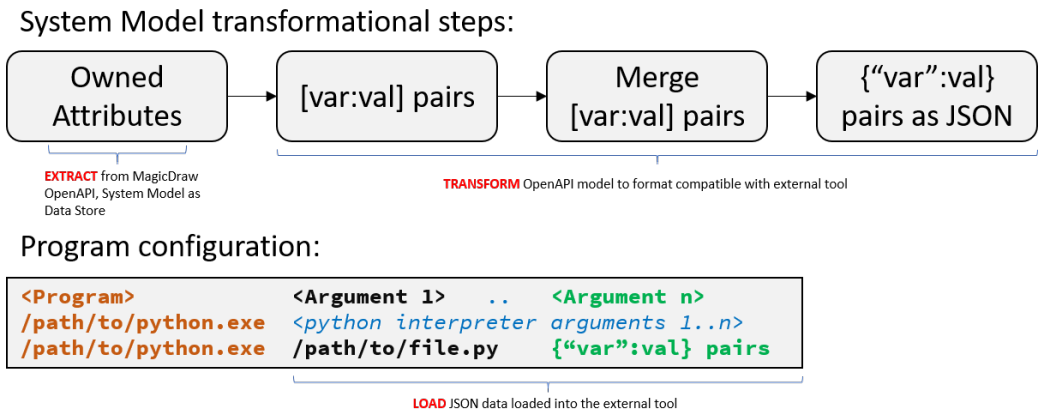


Figure 9.11: Data transformation procedure

This transformation occurs across a series of operations. First of all, the system model is encoded not just in the graphical elements of the user interface, but as a sort of data store via the API of the system modeling tool. This API provides access to information like values according to specific method calls on objects within lists. There is more than one API in the system modeling tool. For running the Walworth model, the standard API as well as a simulation API are in use. In order to make use of the data within the system model, the data must be extracted. The approach here is to construct a projection around the `OwnedAttributes` while filtering out items

which are not of interest in the current application. The excluded content includes constraint properties and ports. Additionally, due to other use cases which arise from time to time, the value extraction enables techniques for handling enumerations separately from typical values. For extracting the actual values, the extraction is simplified by using both APIs in this case as a single method retrieves all values except enumerations. The result of recursive loops over attributes is a datastructure known as a map, dictionary, or associative array, wherein a set of unique keys indicate values. The keys are the names of the attributes — or variables — and the values are the values provided by simulation. This data structure is advantageous for two reasons. The first reason is that it provides an efficient means for linking variable names to variable values, as the time complexity for obtaining a value according to a variable name is usually constant or at worse $O(n)$ for obtaining the values by index of variable name. Furthermore, like many data structures, it can be hierarchical. Consider, the system model, which may have blocks with parts described by blocks with their own parts. Having a data structure with hierarchy enables passing this structure to an external tool, should the tool be compatible with the data. In particular, this might be the `[intensity:0.5]` or similar input to the Walworth model, such that `myMap[(variableName)]=variableValue` for `variableName="intensity"` and `variableValue=0.5`, a relatively natural way to refer to variables and values in configuring the analysis. Unlike the objects which store the data of the system model, this data structure is easily "serialized". Serialization is the conversion of data in the computer memory to a form fit for storage, such as text. A simple serialization is to convert the data to JavaScript Object Notation (JSON), which is a format supported by many programming languages and tools due to its ubiquity in web applications. In this serialization, the appearance and possibly the order of the variable-value pairs may be altered. The example above may become `{"intensity":0.5}`. By using a standard serialization format, data can be easily passed to other environments, e.g.

from a Java-based system modeling environment to a Python code or any other environment. Other options for serialization include XML, CSV, and HDF, but these options may be more or less complicated and have more or less support available by default.

An important aspect of the serialization discussion is that the target program must be able to recognize the data it receives. Somehow, the target program must read the data, parse it, and load it for use. In this case, the target program is the Walworth model implementation in Python which will load the data from the command line as a Python dictionary, and substitute values received in place of default values for solving the initial value problem. A constraint then is that the portion of the system model from which the data is extracted must mirror the structure of information expected by the external tool. The analysis template assists with this, enabling the creation of input and output representations which can capture some template by which the transformation steps described above produce the serialized data compatible with the external tool interface.

One limitation of this approach here should be noted. As implemented, the recursive procedure is only compatible with directed associations in the SysML language. Directed associations are characterized by having only one "navigable" end. Associations with two navigable ends — not having an arrow in their portrayal on diagrams — can result in infinite recursion, as the map bounces back and forth from end to end of the same association. If a model must use relations like these, then an ad-hoc solution or improved and generalized variant of the approach here can be applied. For the purposes of the Walworth model, where the input and output are lists of variables which are scalars or vectors, this concern does not apply and the approach above is valid. The so-called value properties in SysML are defined as directed composition associations targeting value types like `Real` and `Integer`, and there is no risk of infinite recursion.

The outputs of the Walworth model are time series. The command line interface writes the time steps and each primary stock's time series to the command line output (stdout). This data is written again as serialized data in JSON format. The system model which has asked for the Walworth model to run waits for this data to arrive, and then performs the inverse operations as described above. That is, the system model loads the JSON data as an associative array of key, value pairs and then sets the simulation value for each of the time series lists in the output data. To save the data, the simulation values can be written into an instance specification in the system model for later usage, e.g. export to CSV or visualization.

Source code for the two key scripts in the Walworth SysML model integration and simulation are included in Appendix C Section C.1. Specifically, only the scripts which go from SysML to JSON and from JSON to SysML are given; other scripts were implemented by are not included in the appendix as they are more utilitarian and less central to the process.

9.3.2 Integration of the DES Model to SysML

The DES model from Experiment 2 presents a different case for system model integration than the Walworth model. Especially for the simulation of proposed SE tasks, the input should be available in some form of description of the plan. There are multiple ways to describe a plan or process in a system model. One simple approach might leverage blocks or classes as containers for the simulation data required by the DES tool. However, in order to describe a process or sequence of tasks, usually some flowchart capability such as an activity diagram is used. Activities can provide the flowchart depiction while also serving as simple containers themselves for the simulation data. However, in this application the usage of activities has a very different meaning or consequence in describing the SE tasks, versus some other behavior in the system model. It is possible to establish a domain-specific language (DSL), based on

activities, which customizes their appearance, their features, and which may allow the user to specify a task plan that gets written out to the simulation input file format. Figure 9.12 illustrates customizations which are used here to achieve a DSL for SE task simulation.

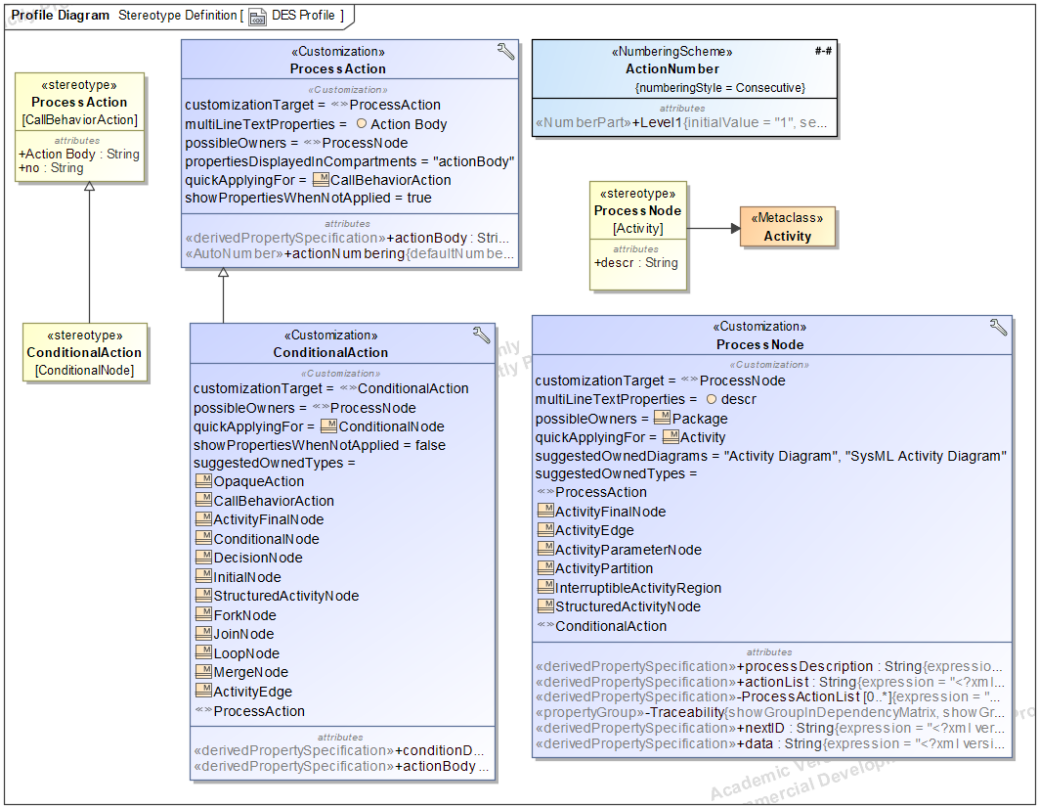


Figure 9.12: Customizations for DES

The approach via Figure 9.12 is a sort of compromise between simpler containers of data, and a highly customized DSL. This lightweight DSL is established which places some limits on activities without changing them too much. Additionally, the activities and actions within the activities have fields specific to the simulation input, and assemble these fields automatically based on the structure of the task plan in the system model. The language has three primary features which restrict the usual SysML syntax for activities. The task is represented as a `ProcessNode`, which actions are represented as `ProcessAction` or `ConditionalAction`. Each of these items places restrictions on what would normally be permitted in SysML. For example, a

ProcessNode can only contain a limited set of diagrams — UML and SysML activity diagrams — and the default call behavior actions are ProcessActions. The main field of ProcessActions are the actionBody. Additionally, the ProcessNode contains other useful fields such as one for the description. One complication is to avoid ProcessNode appearing as ProcessAction when appearing as call behavior actions themselves. To this end the ProcessNodes are joined instead via dependencies to indicate the next node via the nextID property. Data for use in the simulation is conveyed by a use dependency targeting an instance specification. The customizations provide a hook for scripts embedded in the derived properties to generate JSON as an intermediate representation of the simulation data. This JSON is stored in the element specification to be retrieved during simulation as UML properties. The source code for the Process Node Action List and the Conditional Action Condition Description are given in Appendix C Section C.2 as prime examples. A simple example will be used to illustrate the utility of this approach leverage domain-specific language.

Illustrating the Language through a Simple Example

This simple example is meant to illustrate and test the capabilities of the DSL. The ProcessNodes are not representative of actual task flow and reuse descriptive text from the primary method models. The first ProcessNode is meant to be rather simple and apply the uniformAddInt action to some data and generate a passthrough value. The second ProcessNode is meant to be more complicated and exercise the ConditionalAction. Choices for the associated data are aimed at making this setup highly iterative. The overall view can be seen in Figure 9.13, where all portions of the model which exercise the high-level plan are illustrated.

The test plan shows the definition of the ProcessNodes, and how they are connected from P1 to P2 by a dashed line — a dependency. Additionally, the use edge

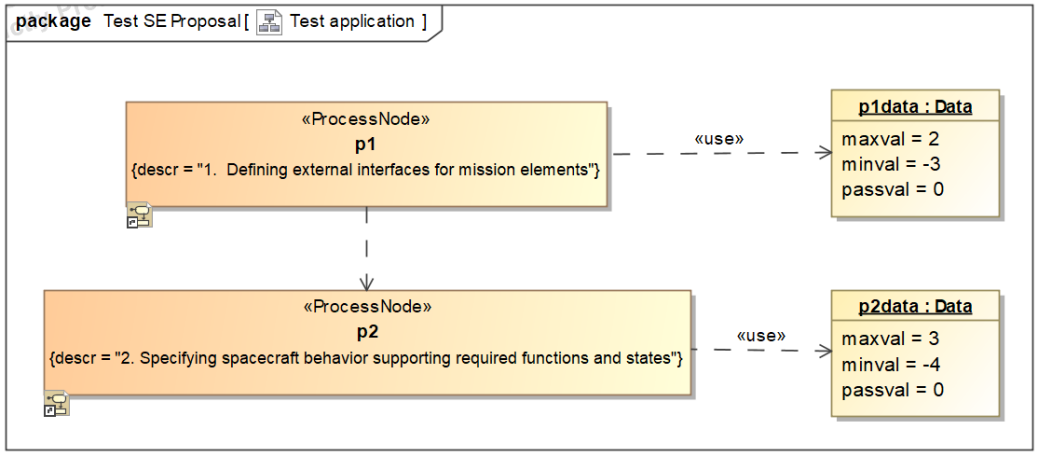


Figure 9.13: Example Setup Plan for DES Simulation

can be seen connected the `ProcessNode`s to their data. Other views expanding on Figure 9.13 are possible, such as Figure 9.14, which presents the information in tabular form.

#	Name	Descr	Next ID	Process Action List	Data
1	p1	1. Defining external interfaces for mission elements	p2	1 print 2 uniformAddInt 3 nextproc	{"maxval":2,"minval":-3,"passval":0}
2	p2	2. Specifying spacecraft behavior supporting required functions and states	null	4 print 5 condition	{"maxval":3,"minval":-4,"passval":0}

Figure 9.14: Example Setup Table for DES Simulation

Figure 9.14 also provides a view into the internals of the `ProcessNode`s. Visible here for the first time is the custom numbering applied to the `ProcessActions`. This numbering is used to sort the `ProcessActions` to ensure that they are transmitted in the correct sequence. Additionally, the computed results for `nextID` from the dependency and `data` from the `use` dependency are visible. `nextID` is plain text that is either the name of the next `ProcessNode` or `null`, while `data` is the content of the instance specification in JSON format similar to that described in the Walworth model transformation discussion. However, as yet, no simulation is applied to the representation, merely computations built into the custom properties of the DSL.

For the example, the first `ProcessNode` internals are illustrated in Figure 9.15.

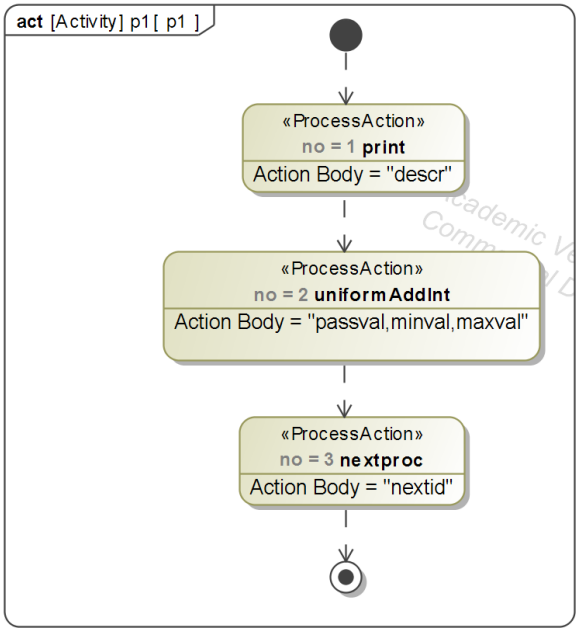


Figure 9.15: Internal description of Example P1 Process Node

The steps within include printing the description, incrementing an integer by a uniform random integer value, and continuing to the next `ProcessNode`. The description of this node is reused for demo purposes. Clearly, there are two levels of detail which will interest different stakeholders. The outer view of the `ProcessNode` network is more interesting to planners and managers. However, the internal view is where the details relevant for simulation in terms of aspects of duration, state variables, or other quantities can be specified. These details drive the simulation behavior, and must be populated in order for the generated input file to be compatible.

Figure 9.16 illustrates the second example `ProcessNode` internals. This example `ProcessNode` tests the `ConditionalAction` properties. For the `ConditionalAction`, the details should be specified in a format similar to that shown in Figure 9.16. Note, that for this rapid implementation (time to implement was approximately 1 week), there are no constraints or validation rules for the formatting of the data in the various possible fields. Thus, in reusing the information here, the DSL must be applied with care to replicate the format of the data. The rules for the `ConditionalAction` include:

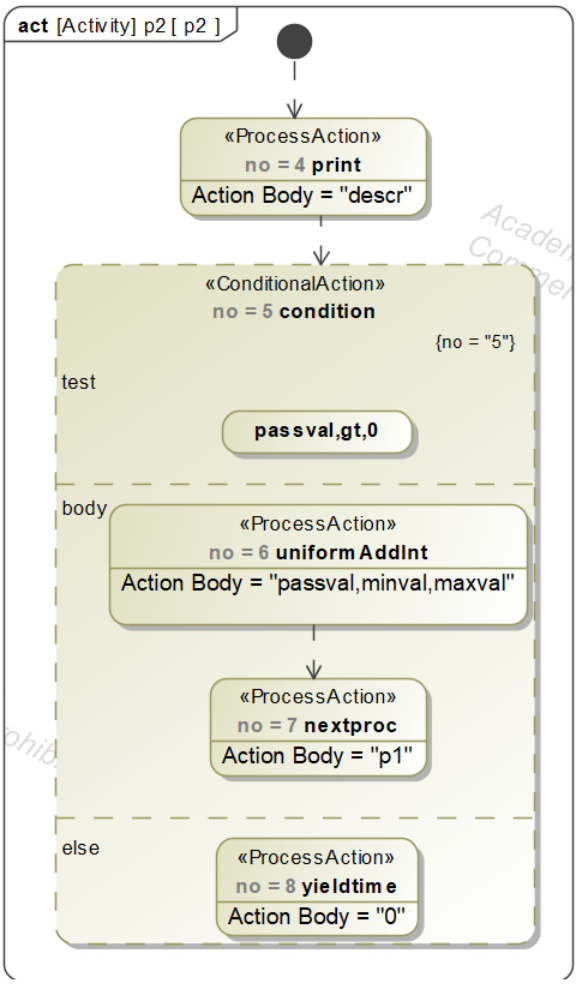


Figure 9.16: Internal description of Example P2 Process Node

1. The `ConditionalAction` shall be named “condition”

2. The test shall contain one opaque action.

   - The test opaque action shall include 3 comma separate values
   - The first value shall be the data variable used in comparison
   - The second value shall be the Python operator shorthand for the comparison
   - The third value shall be the quantity against which the comparison will be made

3. The body shall contain 1..n `ProcessActions`, terminated in a `nextproc` whose body indicates the node to which the iteration returns

4. The else shall contain a `ProcessAction` to `yieldtime` or `nextproc` if a `nextID` is available

The implementation here is a sort of minimum viable product which enables authoring the method model and generating the simulation. A more-user-friendly implementation would help to enforce and assist in conforming to the rules enumerated above. However, given that the information is correctly populated, it is possible to then configure an analysis.

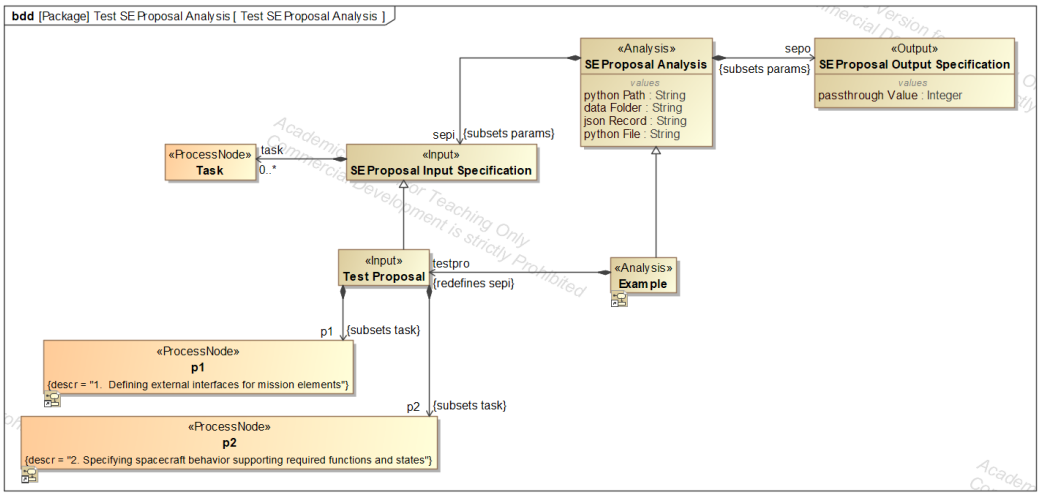


Figure 9.17: Analysis Configuration for DES Example

Figure 9.17 illustrates the configuration of an analysis for running the example method model through the simulation. The input has two properties defined by the `ProcessNodes` of the method model. These properties subset a single inherited property `task` for ease of coding. The entire simulation is then driven by a script embedded in the behavior for the analysis element, and the results will be shown for the subsequent method models of interest.

SRD Method Model

The SRD Method Model is the primary model of interest, simulating the SE task proposal from Friedenthal and Oster[98]. In SRD, the spacecraft requirements are derived from mission requirements, according to analytical results. Each task proceeds one to the next in a linear fashion. Figure 9.18 illustrates the SRD method model implementation using the DSL presented above. Each task, represented by a `ProcessNode`, `use`s the same `data`. This `data` is used to represent a duration for the `ProcessNode` in the case of this simulation; that is, the meaning attached by the user of the simulation to this value might be the duration, although such meaning is not enforced strictly by the simulator tool, as discussed in Experiment 2.

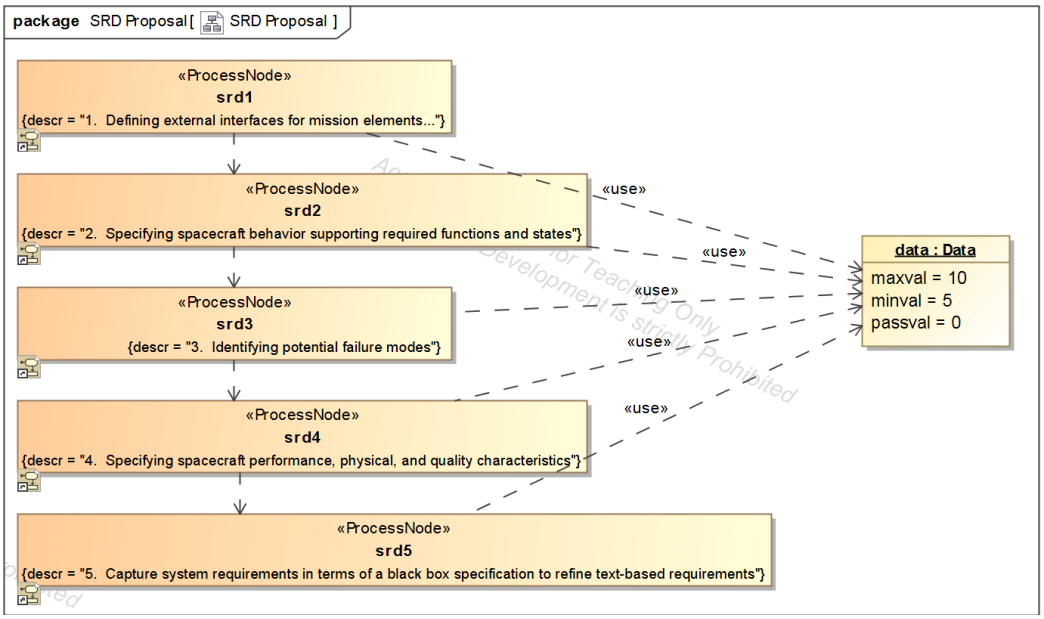


Figure 9.18: SRD Proposal Setup

In the SRD method model, the first four `ProcessNode`s are identical. They each will print their description, increment the passthrough value by a uniform random integer to represent their duration, and then transition to the next `ProcessNode`. Their internals are illustrated in Figure 9.19. It is possible to copy and paste `ProcessNode`s to accelerate the specification of the method model. In these tasks, the primary pur-

pose of simulation is to check the variability in the duration of the SRD method. For this purpose, it may be useful to have separate `data` for each `ProcessNode`, and that can be configured by the `use` dependency. However, in the `ProcessNode` internal view, all that is visible are the `ProcessActions`, and updated `data` would not be visible from the internal view. The final `ProcessNode`, here labeled `srd5`, is different

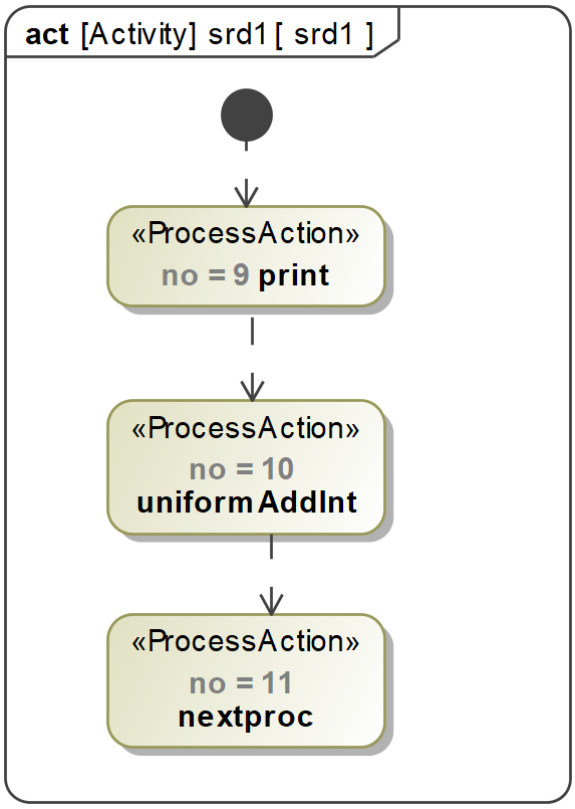


Figure 9.19: SRD Process Nodes 1-4 Internals

insofar as it concludes with the necessary `yieldtime` of 0 `ProcessAction`. The internals of `srd5` can be seen according to Figure 9.20. As a reminder, the `yieldtime` of 0 is necessary to provide the library used by the simulator a means to generate effectively a null event which concludes the simulation, as every event must be compatible as a Python generator. However, it should be possible to also use the `yieldtime` `ProcessAction` to insert durations of fixed value into the simulation. Despite this potential capability, there is not currently any linkage between the arguments for `yieldtime` and the `data` variables such as the passthrough value `passval`. Bringing

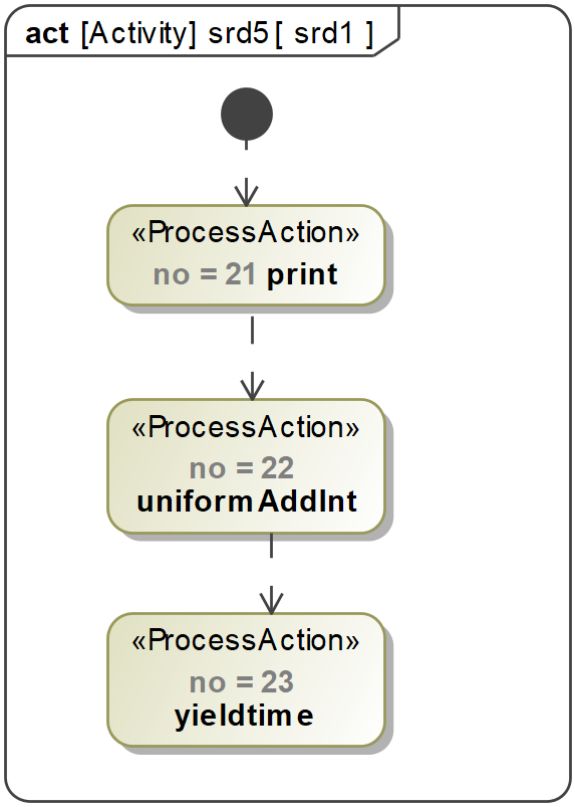


Figure 9.20: SRD Process Node 5 Internals

these ProcessNodes together into the overall plan is Figure 9.21, which presents the information in a table similar to the example. The table view presents a convenient summary for reviews and decision-making purposes, in terms of what ProcessNodes are specified but also the data used for them as far as whether the ranges are reasonable for minimum and maximum values. Additionally, the ProcessActions within the ProcessNodes are visible and sorted according to their numbering. Note that the numbering implementation is currently very simplistic. The numbering is a global count of all ProcessActions. The main utility of the numbering is to order the ProcessActions within a method model. The table in Figure 9.21, however, is sorted by name of the SRD ProcessNodes.

The description becomes more interesting with the SRD Analysis illustrated in Figure 9.22. Apparent in Figure 9.22 are several aspects of reuse. As in the example implementation, the input consists of properties using the ProcessNodes and subset-

#	Name	descr	Next ID	Process Action List	Data
1	srd1	1. Defining external interfaces for mission elements...	srd2	9 print 10 uniformAddInt 11 nextproc	{"maxval":10,"minval":5,"passval":0}
2	srd2	2. Specifying spacecraft behavior supporting required functions and states	srd3	12 print 13 uniformAddInt 14 nextproc	{"maxval":10,"minval":5,"passval":0}
3	srd3	3. Identifying potential failure modes	srd4	15 print 16 uniformAddInt 17 nextproc	{"maxval":10,"minval":5,"passval":0}
4	srd4	4. Specifying spacecraft performance, physical, and quality characteristics	srd5	18 print 19 uniformAddInt 20 nextproc	{"maxval":10,"minval":5,"passval":0}
5	srd5	5. Capture system requirements in terms of a black box specification to refine text-based requirements	null	21 print 22 uniformAddInt 23 yieldtime	{"maxval":10,"minval":5,"passval":0}

Figure 9.21: SRD Proposal Table Summary

ting `task`, and belonging to a dedicated `Analysis` block. While the behavior is very similar to that of the example, it is not fully reused as some aspects are hard-coded into the implementation for speed of implementation. The analysis configuration reads in the data from the `ProcessNode`s themselves, and configures it for the simulation tool similar to the Walworth model description. However, in this case, a

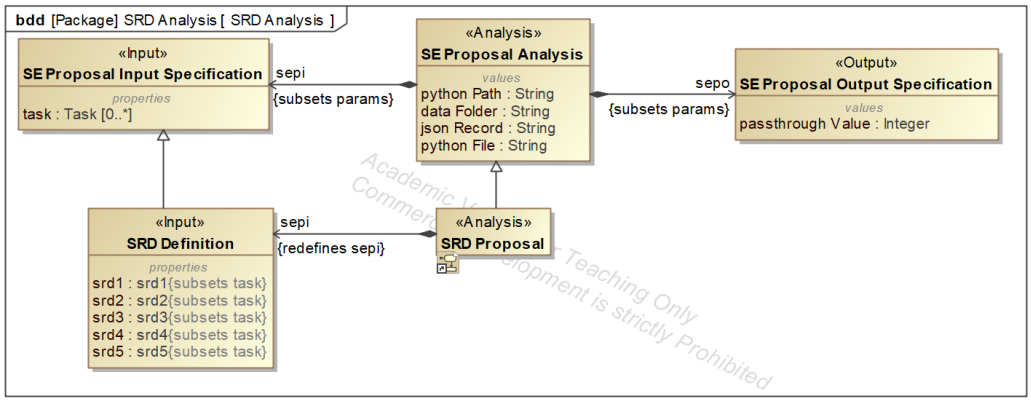


Figure 9.22: SRD Analysis Configuration

singular JSON object is constructed, with keys according to the shorthand identifiers of the `ProcessNode`s, and the content for each `ProcessNode` being the description, the action list, the `nextID`, and the associated data. As this information is already available in the system model due to the DSL customized properties, these properties are merely read from the `ProcessNode` elements keyed by the input block properties. In this way, creating the SRD plan using this DSL automatically creates all the data

necessary to write the input file for the Experiment 2 simulation environment.

Figure 9.23 illustrates the results returned to the system model. Additional code could deliver these results to the output for storage, as in the Walworth implementation. However, that coding step is taken here as trivial given the demonstrations up to this point. The only potential complication in reading this data back to the system model might be the iteration parsing for SAMD, which is discussed in the subsequent as the primary alternative method model.

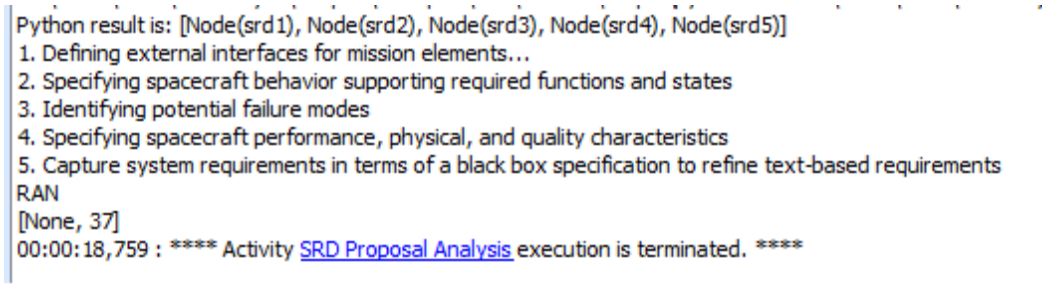


Figure 9.23: SRD Result in CST Console

SAMD Method Model

SAMD is the primary alternative method model under consideration. As a SE method, it is substantially more complex than SRD. This additional complexity is due to the need for potential iterations in model development on newly discovered state variables. Furthermore, it is the reason for the example to use the `ConditionalAction`. A final layer of complexity is visible in Figure 9.24, in that the data vary from `ProcessNode` to `ProcessNode`. Figure 9.24 illustrates the setup diagram which parallels those shown earlier for the example and for SRD. SAMD has seven `ProcessNode`s but some nodes share the same data, while others have their own data. The initial value for the simulation is provided by the first `ProcessNode`, which starts with a passthrough value of 10 used here to represent 10 state variables. This initial condition is easily modified by changing the slot value for the `samd1 Data`, and the subsequent minimum and maximum values are also modifiable in a similar manner,

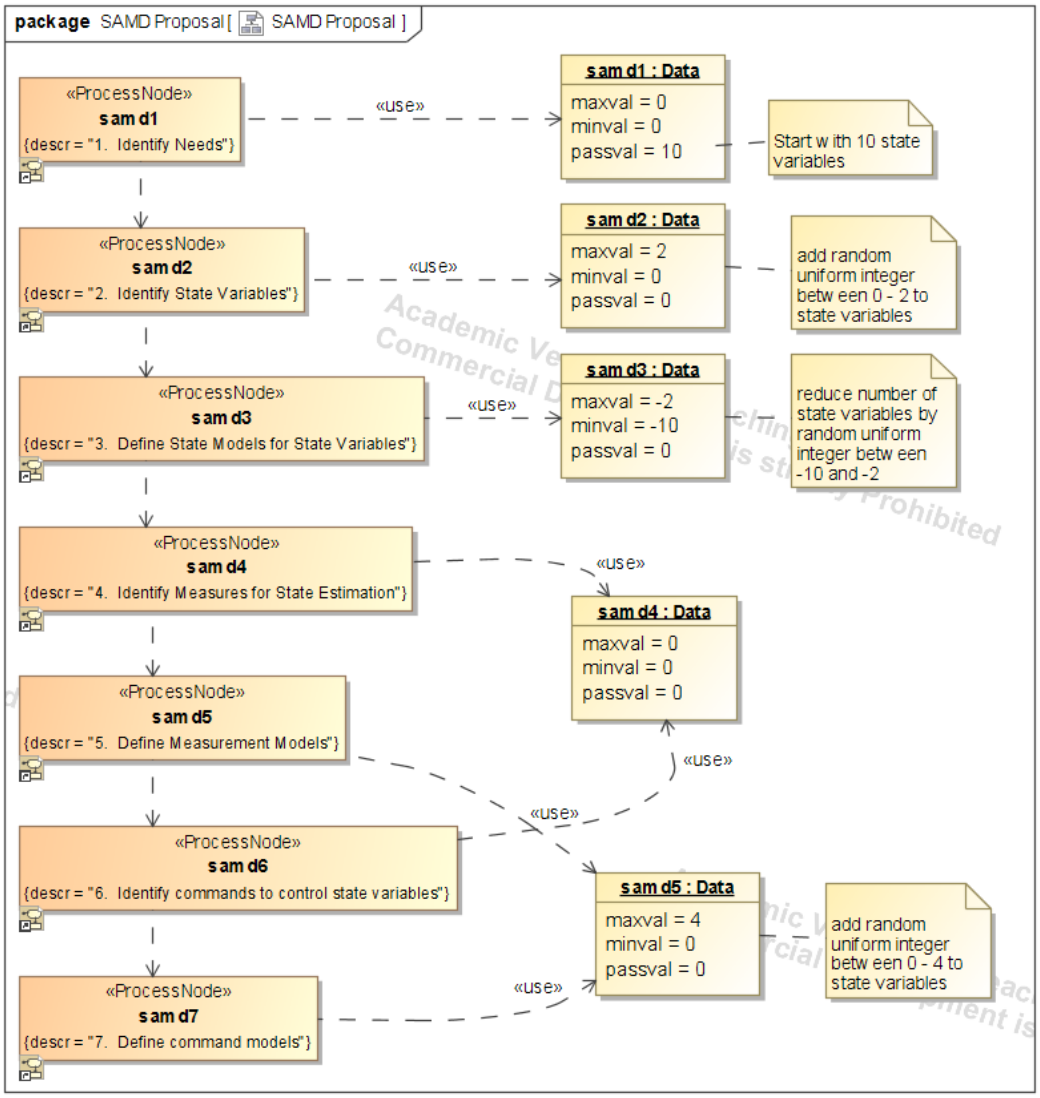


Figure 9.24: SAMD Proposal Setup

enabling exploration of different simulation outcomes via the DSL. Note that because the iteration is conditional on the value of the passthrough, it is possible to configure the data such that the simulation has an infinite loop. Keep in mind for these cases that the comparison must eventually terminate the simulation.

However, despite greater overall complexity, some `ProcessNode`s in SAMD are much simpler than before. These simple nodes are illustrated by Figure 9.25 and include nodes 1, 4, and 6. These nodes do not describe any changes to the passthrough value, state variables, in the method proposal. Therefore, in this representation, they

merely print their description before proceeding to the next task. A more complex implementation of the simulation environment, as discussed in Experiment 2, might implement a full parser-interpreter on the input data. This implementation might have better capability to consider multivariate nodes. If nodes had multiple variables

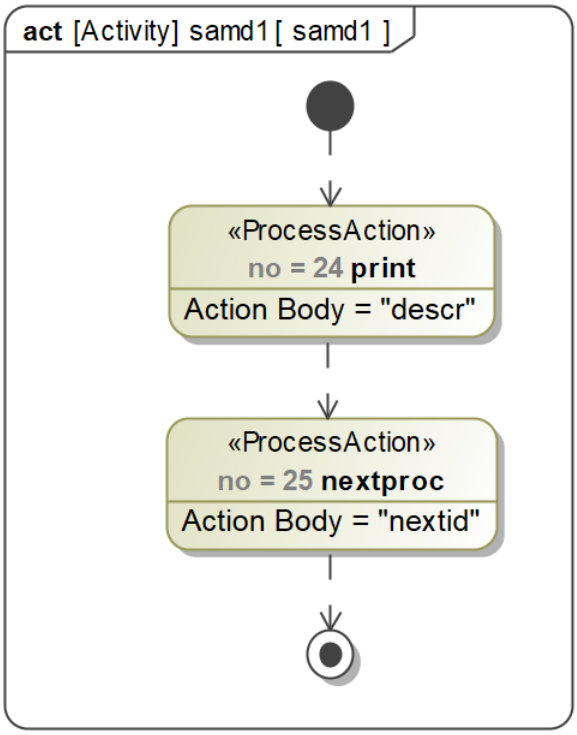


Figure 9.25: SAMD Process Node 1, 4, and 6

with or without bounds like the passthrough variables, then these "lame duck" nodes might yet serve a purpose in modeling overall plan duration. However, as implemented without a full parser-interpreter, the environment has limitations on its ability to read the input file and in this case, the limitation is on the modelers interpretation of how to use the data associated with each node. In this case, the singular passthrough value cannot be used for duration.

The core `ProcessNode`s of SAMD are nodes 2, 3, and 5. The body of these nodes is similar to what was seen for SRD, and is illustrated in Figure 9.26. Figure 9.26 shows that these nodes do modify the state variable count according to their data. In this case, depending on the data, the nodes may increase or decrease the amount of state

variables. While similar to the behavior of the passthrough value before for SRD, this representation takes the neutral nature of the passthrough and considers it to represent state variables instead of duration. This is due to discovered/un-modeled state variable count being the primary measure of the SAMD process which determines its evolution. For SRD, there is no equivalent, and thus the passthrough is used to represent duration in the interpretation of the simulation results. The `ConditionalAction`

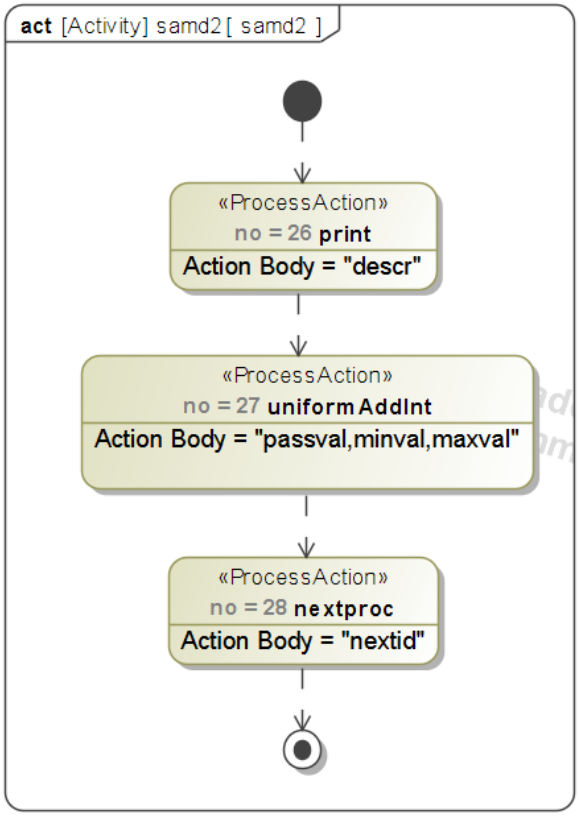


Figure 9.26: SAMD Process Node 2, 3, and 5

of SAMD is the key feature of this method proposal. The iterative behavior of the method is determined by the condition posed by the `ConditionalAction` and the node which the process then revisits. For the `ConditionalAction`, the features of the UML conditional structured action are formatted to include `ProcessActions` and the expectation is that the test will contain comma separated values, as illustrated in Figure 9.27. The action body of the `nextproc` must be the name of a

ProcessNode which the simulation will revisit. While this is similar in appearance to the ProcessNode nextID, it is not at this time automatically populated, so care must be taken in specifying the name. Note also that the simulation environment will expect the ConditionalAction to be named "condition" in the assembly of the action list for the ProcessNode. As described earlier, different operators are possible to use in the test, however the format must be compatible with the Python operator library. The full proposal setup for SAMD is illustrated in Figure 9.28, which presents the

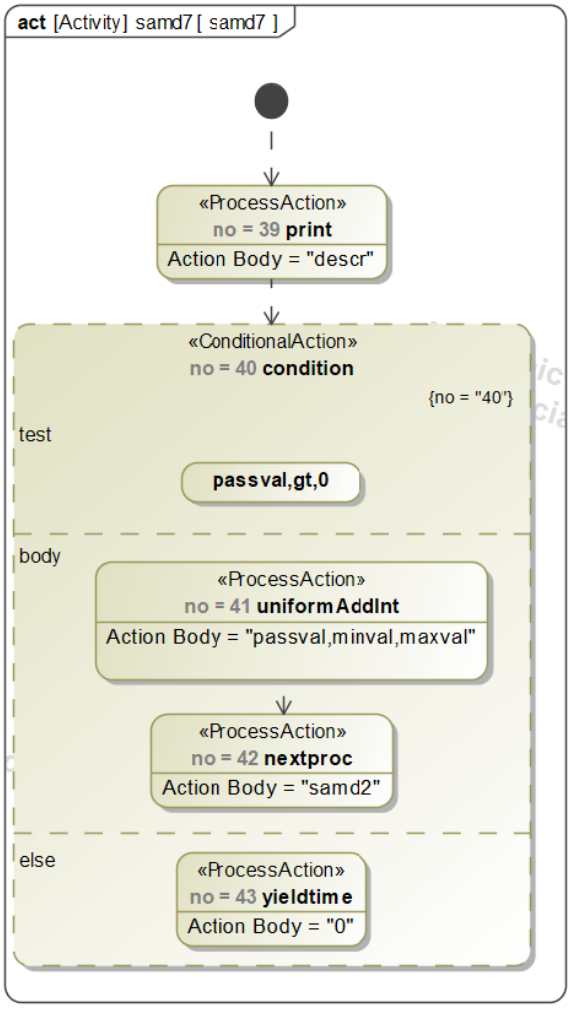


Figure 9.27: SAMD Process Node 7 with conditional behavior

tabular view. In this view, each action in the ProcessNode action list is ordered by number but again the numbering is global for the system model, not for the method

proposal in particular. The data can be viewed here, but edited on the first diagram which lays out the proposal. If satisfied, the next step is to construct the analysis and run the simulation environment with the method proposal input information. The

#	Name	Descr	Next ID	Process Action List	Data
1	samd1	1. Identify Needs	samd2	24 print 25 nextproc	{"passval":10,"minval":0,"maxval":0}
2	samd2	2. Identify State Variables	samd3	26 print 27 uniformAddInt 28 nextproc	{"passval":0,"minval":0,"maxval":2}
3	samd3	3. Define State Models for State Variables	samd4	29 print 30 uniformAddInt 31 nextproc	{"passval":0,"minval":-10,"maxval":-2}
4	samd4	4. Identify Measures for State Estimation	samd5	32 print 33 nextproc	{"passval":0,"minval":0,"maxval":0}
5	samd5	5. Define Measurement Models	samd6	34 print 35 uniformAddInt 36 nextproc	{"passval":0,"minval":0,"maxval":4}
6	samd6	6. Identify commands to control state variables	samd7	37 print 38 nextproc	{"passval":0,"minval":0,"maxval":0}
7	samd7	7. Define command models	null	39 print 40 condition	{"passval":0,"minval":0,"maxval":4}

Figure 9.28: SAMD Proposal Table Summary

analysis configuration is identical to the previous models. As in the example and in SRD, the SAMD proposal analysis uses an input block with properties defined by the `ProcessNode`s. The system model data is parsed out to the analysis by the behavior within the SAMD proposal analysis. The overall configuration is shown in Figure 9.29, including the reuse of the task list definition and subsetting for use of the list of `ProcessNodes`. If defining a new output block, this analysis would need to define an additional value for the number of iterations, beyond just the final quantity in the passthrough value. Additionally, the passthrough value for the SAMD proposal may need to be redefined for multiplicity according to the values updated across iterations. Figure 9.30 illustrates the result in the simulation console as well as the script which manages the system model parsing and running of the analysis. This behavior is largely reused from the SRD implementation. A more modular setup is possible, but in the configuration effort described in this chapter the script itself is not the most challenging setup — more likely are typos or other issues which spoil the configuration file generated from the method model applying the DSL. While debugging these errors can be challenging, the end result when properly configured

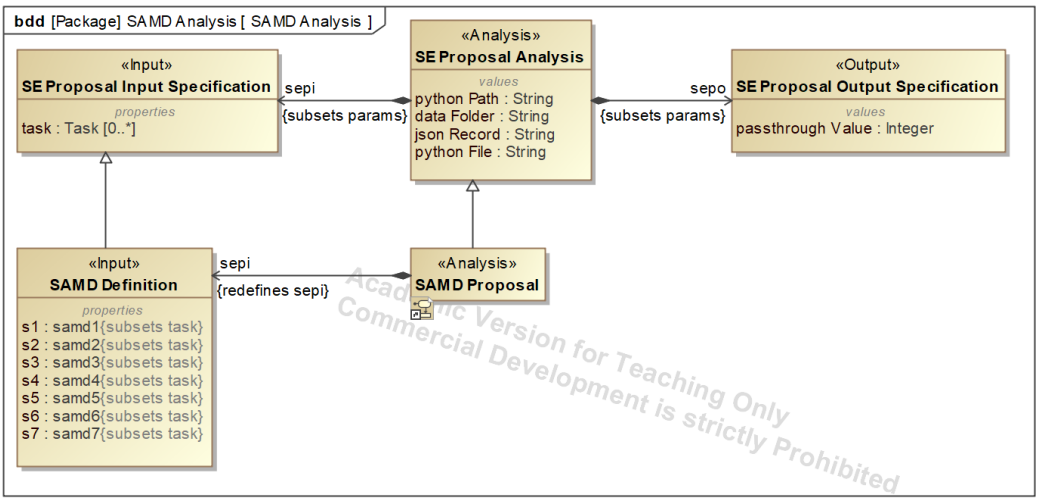


Figure 9.29: SAMD Analysis Configuration

exactly matches the inputs from Experiment 2 while providing a graphical means to specify the method models in the system model for SE planning purposes.

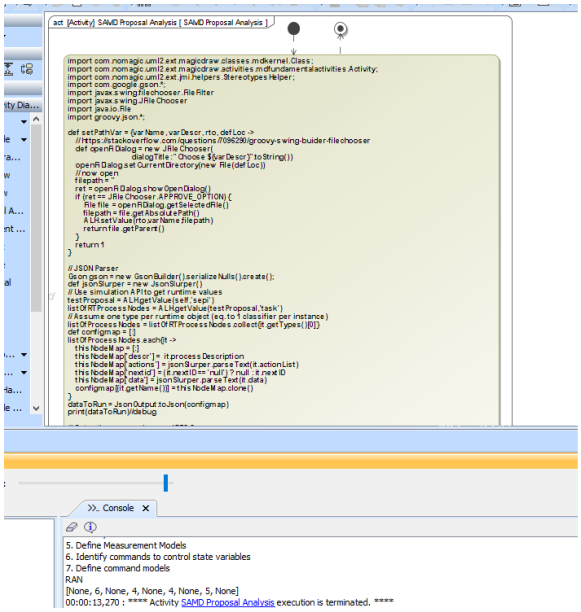


Figure 9.30: SAMD result in CST console

9.4 Conclusion

Despite the shortcomings of Experiment 3a, Experiment 3 shows much success via Experiment 3b and the specification of method models in the system model for an-

alyzing the SE method proposal. While the Walworth example is typical of more routine analyses, the DSL constructed for the DES representation is a potential contribution of this work and provides another means for simulation. Beyond the specific target applications of SE planning, other DES using the actions posed by Experiment 2 may be possible via the system model, and most importantly other than the system modeling tool itself, the core simulation environment is constructed using open-source software. Thus, while many similar techniques are significantly locked down, this technique may be extensible to other system modeling alternative environments. Insofar as the DSL implementation illustrates the promise of system model customization for domain analysis, Experiment 3 illustrates the practical capability of these languages beyond representing semantic information underlying SE documentation. While Hypothesis 3 is for the moment rejected due to the lack of bi-level simulation available in Experiment 3a, environments such as that constructed here in Experiment 3b answer Research Question 3a by enabling the exploration of parameter values in close coordination with the system model environment, thus enabling better exploration of the process, schedule, or other design space as documented and managed within the MBSE ecosystem.

10

EXPERIMENT 4: EXPLORATION OF MODEL PARAMETERS

10.1 Recap of the Experiment

Experiment 4 seeks to address Hypothesis 1 with the completed environment. Hypothesis 1 was defined as:

Hypothesis 1 If a model for a SE measure predicts progress in a SE process in correlation with a lagging indicator such as cost, schedule (often combined as effort), etc, then the model provides a basis for decision making on the method for a SE process.

Given the capabilities proposed over the previous experiments, it would now possible to investigate how the leading indicator responds to the task graph and to understand the impact of planning choices. With Experiment 4, the full capability to explore method proposals would be available. Future work could attempt to provide concrete correlations between the leading indicator and cost models, a missing functional relationship that impedes getting costing numbers in terms of the SE tasks. The platform enables the consideration of new SE methodology proposals in terms of their impact on a leading indicator trajectory, i.e. an s-curve, at least in the case of requirements volatility during early-phase validation actions. Experiment 4 proposed the following method:

Experimental Procedure :

- Parameterize the system model representation
- Use the system model representation to generate simulation models

- Evaluate the simulation models over a design of experiments
- Fit surrogate models to the leading indicator measure
- Use the surrogate models of the leading indicator to explore the SE method design space

10.1.1 Revised Procedure

Originally, Experiments 1, 2, and 3 were planned to be primarily focused on model-building, while Experiment 4 aimed to focus more on model exploration. However, in the course of Experiments 1 and 2, some of these capabilities were developed and exercised on the models in question directly. Specifically, a key finding from Experiments 1-3 is that the intended leading indicator model, Walworth et al, cannot be properly integrated with the DES-based method models at this time. Additionally, surrogate modeling has falling outside the scope of the work for the time-series data.

The integrative nature of this experiment, however, can still be realized. Several aspects of the previous experiments remain to be unified, and there is an opportunity to look ahead towards Experiment 5. Furthermore, new concerns have arisen, namely a need to perform a direct comparison of named SE methods, where as originally the plan was oriented towards variation of a singular process. These combined needs drive a re-structuring of Experiment 4 towards answering Hypothesis 1. Importantly, the deficiencies in models discussed previously do not necessarily reduce the truth value of Hypothesis 1. Experiments 1 and 2 presented model formulations which illustrate the completion of SE work according to various organization, project, and process variables and result in some measure of the amount of time either directly, abstractly, or indirectly by number of iteration cycles. This measure of duration of the work correlates to cost, as discussed in Experiment 2. Therefore, exploring the variability of these models may still enable decision-making on SE methods for SE processes.

To further explore these issues, Experiment 4 will follow a revised method.

Revised Experimental Procedure :

- Summary of Parameterizations Established
- Specification of Cases in System Model
- Establish Additional Criteria for Methods
- Establishing a 3rd Method Model
- Consideration of Cost
- Direct Method Comparison

10.2 Parameterizing the System Model

The parameterization of the Walworth model in the System Model is straightforward:

$$W_{p,i} = \text{Walworth Input Paramters}$$

$$W_{p,o} = \text{Walworth Output Parameters}$$

There are a set of inputs and a set of outputs. These are explicitly modeled as discussed in Experiment 3. However, more difficult to understand is to what extent the DSL of Experiment 3 for DES is parameterized. Fundamentally, the problem is one of exploring the **architectures** for the method models. This topic is complicated, as in Kerzhner [121], Iacobucci [117], and Sharma [122], who all posited the use of a combination of discrete and continuous variables in their exploration of architecture. In the DSL of this thesis, continuous variables include items such as the initial passthrough value, minimum, and maximum in the data for a `Process Node`. Discrete variables represent the graph structure underlying the DSL: how many different data elements are used, how many `Process Node`s are used, which and how many actions, and does the method iterate — all this must be considered.

Architecture Parameterization per Kerzhner

Kerzhner [121] bases the formulation of architectural variables on disjunctive programming, citing Grossmann's extension to mixed-integer programming[123]. Specifically, this is a technique by which groups of discrete and continuous variable coexist, and where boolean variables "control the part of the feasible space in which the continuous variables, x, lie"[123, p. 242]. Kerzhner adapts this technique, and claims to reduce the the number of such variables, while asserting "these binary variables represent all potential components and connections" in the architectural description[121, p. 158]. However, these binary variables describing the content of the architectural description are not explicit; instead, "the binary variables related to the selection of a particular architecture are implicitly defined through the use of connection templates and multiplicities in the original SysML model"[121, p. 185]. Therefore, no explicit definition of the binary variables is provided, although the algorithm which performs a series of transformations in Kerzhner's implementation does at some point derive from the architectural description the set of binary variables for the input description. That said, no explicit representation is made in the architecture or ahead of parsing the architecture for transformation, and this is an important result. Instead, generalized portions of the modeling language used for architectural description are interpreted such that the variables can be established by the program. Additionally, binary variables are also used to parameterize behavior, for example in the representation of state such as on and off[121, p. 224]. Kerhzner gives an example of a specific subset of the hundreds of binary variables in his problem. This example focuses on the selection of a pump variant from a set of four possible pumps: (x_1, x_2, x_3, x_4)[121, p. 266]. This set takes on a boolean value of $(0, 1, 0, 0)$ if x_2 is selected, and may represent the selection of multiple options by increasing the number of 1s in the set. However, this is discussed by Kerzhner in the context of multiple different combinations of binary variables which together represent identical architectures, per the

binary variable formulation. After applying some technique to filter these options, Kerzhner succeeds in formulating a mixed-integer type problem using mathematical programming to explore architecture and design optimization in terms of the discrete and continuous variables.

Architecture Parameterization per Iacobucci

The technical problem approached by Iacobucci is very similar to Kerzhner, yet this similarity is not readily apparent due to the differences in application domain and solution procedure. While Kerzhner advances QVT and Java tools for modifying the SysML description and establishing transformation rules to simulation programming languages, Iacobucci dismisses SysML as pure "marketecture"[117, p. 65] and seeks a different means of representing and exploring architectural performance. While both Kerzhner and Iacobucci would agree that mathematical models for performance evaluation should be simplified during the search of the architecture space, Iacobucci's approach is not to apply a constrained optimization technique so much as to exhaustively enumerate the architecture alternatives. Iacobucci achieves this using functional programming constructs and by creating a DSL in Lisp which both represents the architecture space, generates architecture alternatives, and applies the concepts of Map-Reduce to roll-up the evaluation of architecture performance in search of better architectural solutions[117]. The different application domains of these theses obscures similarity: Iacobucci seeks to perform system and operational trades by generating and executing architecture models[117], and effectively is searching for "better" architectures — in the context of the Department of Defense Systems of Systems approach, rather than in the context of "architecture" as discussed by Kerzhner, though both usages are compatible under ISO 42010[13]. It is merely the kinds of architectural descriptions which vary, not the facts of architecture nor the problem of exploring the space. Similar to Kerzhner, the variables which describe

the constitution of architecture in Iacobucci are not explicit. Instead, the first phase of deploying Iacobucci's solution is to formulate the problem in his DSL, describing various spaces — capability hierarchy, candidate systems (e.g. similar to pump variants, but "vehicles"), and finally a computer model for the "System of Systems" [117, p. 82]. Iacobucci's computational models transform "the system level scores into architectural alternative level scores"[117, p. 87]. The roll-up of scorings provides the basis for an objective function on the architecture space which is appropriate for the domain of application. After exhaustively generating all architecture alternatives and rolling up different metrics, Iacobucci applies Multi-Objective Decision-Making on the architecture space according to the rolled-up metrics. One key difference between the two approaches other than in search methodology is in the objective function formulation, where Kerzhner applies cost as an objective function instead of holding multiple metrics as co-equal; yet Iacobucci does also leave room for combining all metrics as an Overall Evaluation Criterion.

Architecture Parameterization per Sharma

Finally, due to the spacecraft-related domain of the current document, the architecture exploration approach in Sharma[122] should be considered. Sharma takes a different perspective to Iacobucci and Kerzhner. The approach in Sharma is narratively defined by the Matrix of Alternatives, Axiomatic Design, and set-theory. Sharma in particular is interested in space mission architectures, those being some combination of vehicles, destinations, trajectories/missions, etc, which again are very different domains of architectural description than Iacobucci and Kerzhner. However, there is agreement as to the discrete nature of architecture. As in the definition of graphs, Sharma takes a set-based approach to defining architecture, design, and objective space relations, by establishing mappings between subsets of architectural, design, and objective spaces. Figure 10.1 illustrates the similarity between the vision

of Sharma and Kerhzner in the visualization of the problem. Specifically, Figure 10.1a

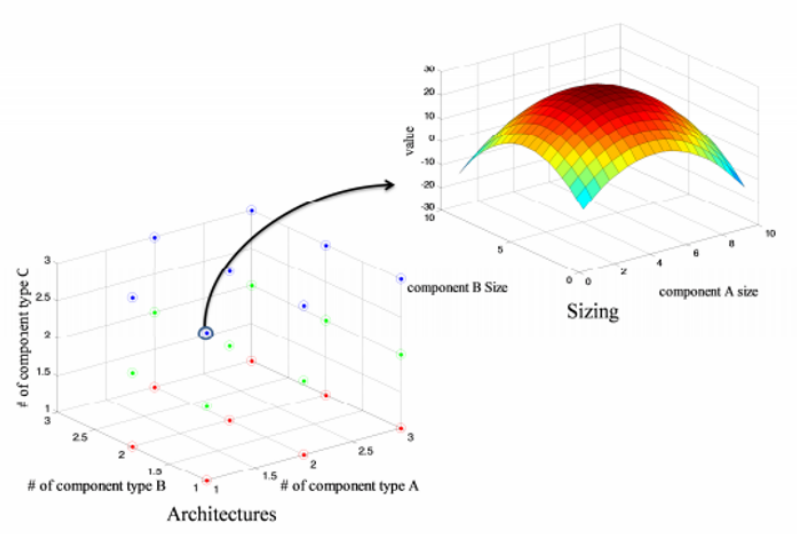


Figure 5.1: Visualization of the design space.

(a) Kerzhner's depiction of Architecture and Objective Spaces[121]

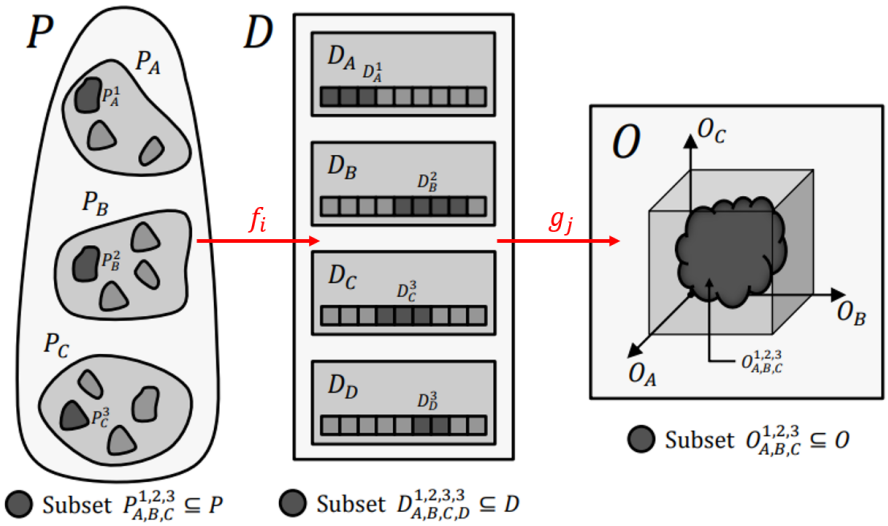


Figure 54: Notional System Architecture $\hat{A}^{1,2,3}_{A,B,C}$ Across the Set Theory Representation of the System Spaces

(b) Sharma's spaces, overlaid with maps[122]

Figure 10.1: Sets, spaces, and mappings for architecture exploration

is Kerzhner's representation and Figure 10.1b is Sharma's representation. Notably, however, Sharma permits for the possibility of multiple vectors of design attributes

per architectural option (e.g. if pump alternatives where characterized by vastly distinct pump technologies), as an intermediate step, while Kerzhner infers directly all design variables used in the objective function formulation based on the architecture option. This difference is notable in the sense of perspective; it may not contribute much to the technical outcomes. This difference originates due to the lack of architectural description language applied by Sharma. While an approach like Iacobucci's might supplant the Matrix of Alternatives in generating alternatives automatically, Sharma uses the Matrix of Alternatives to represent the architecture options and the analysis proceeds after either one or more selections from this matrix[122, p. 136], or by generating a filtered list of the options from the matrix itself[122, p. 151]. Design space variables are optimized for cost and payload mass as multiple objective functions, and the pareto frontiers are rolled up according to the architecture variables. While architecture is handled exhaustively, optimization techniques are applied to the design attributes linked to the architectural selection according to the mappings by genetic algorithm for the pareto frontier search. Unclear is whether the architecture-design mappings could have been systematized further as functors, for the image of the architecture subspace in to the design space. An important difference between Sharma and both Iacobucci and Kerzhner is that Sharma explicitly lists architectural parameters rather than defining the alternative space by DSL; this is because the parameters are necessary as columns for example in the Matrix of Alternatives. As expected from Kerzhner and Iacobucci, the parameters in Sharma are not generalized for architectures more broadly, but specific to the class of architectures studied. These parameters for architectures in Sharma include Destination, Duration, Insertion Mass at Low Earth Orbit, Destination ΔV, Return ΔV, Number of stages, and propulsion technology for stage x[122, p. 139]. Key differences here are that since the Matrix of Alternatives approach is employed, existence assertions are not captured in the same manner: presumably the "number of stages" keys into incompatibilities for second

stage options should the number of stages be only one[122, p. 124], excluding them from selection. There are no binary variables as such representing the existence of one or two or more stages, but instead a binary compatibility assertion between the options in the Matrix of Alternatives.

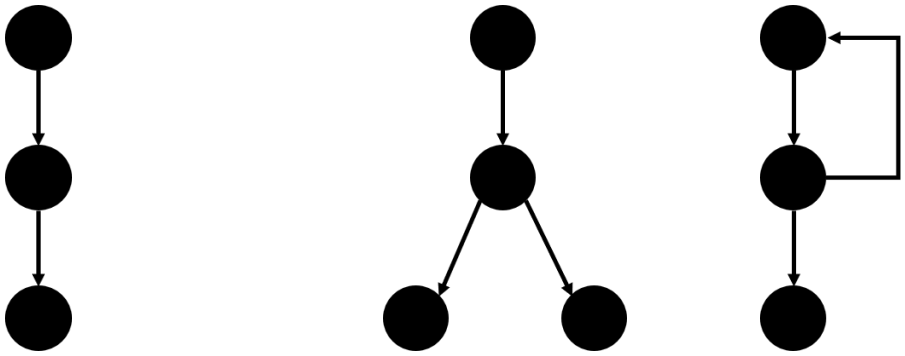
Candidate Parameterization for the Experiment 3 DSL

The bottom line is that some parameters must be defined to assist in exploring SE method models. From Experiment 2 and 3, the primary definition is the input specification, which is in the form of the DSL. At first, it may seem reasonable to follow the path of Kerzhner or Iacobucci, since they also began with DSLs. However, the use of the parameters is also important. The objective here is not to find the best SE method, not to explore all SE methods exhaustively — the data in Experiment 2 for the error detection model showed that perhaps keeping things simple may be best. Instead, parameters are needed which enable comparison between the methods based on their characteristics. This comparison might be aided in part by a Matrix of Alternatives and Compatibility Matrix as in Sharma, but need not be for the purpose of exhaustive generation of alternatives, as that purpose was to find a pareto frontier of optimizing architectural options. Importantly, there is a significant degree of freedom available for the definition of "architectural parameters".

First of all it is important to clarify terminology. The term architecture is vague. Architecture means very different things to each of Kerzhner, Iacobucci, and Sharma, even though there are basic facts in common to the problems they solve. According to ISO 42010, a system's architecture is the "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution"[13]. Commonly it is understood that any real product has an architecture which gives rise to its behavior; in design, the work is to create a description of the real architecture sufficient for characterizing the

intended behavior. To this end ISO 42010 defines an architecture description as a "work product used to express an architecture"[13]. Consider for a moment the work products at play here. Kerzhner's transformation rules rely on QVT, Query-View-Transformation, which is a language that affects *graph transformation*, taking in a *graph* and giving out a graph according to rules which map from one metamodel to another as managed by Java code. Iacobucci requires a capability hierarchy, *a form of graph known as a tree*, to be defined using his DSL. A graph is defined according to sets of nodes and edges, where each edge is a set of two connected nodes. A tree is a kind of graph where each there are no cycles, or paths, which return to previous nodes, thus any two nodes have one path of edges which connect them. Further more, while Sharma does not work explicitly in graphs, Sharma does work in sets, which are the building blocks of graphs. Sharma abstains from cohesive architectural description, using just a matrix of alternatives, but this representation may be shown according to graph-like languages such as SysML — Nelessen[124] does this for generating alternatives. Moreover, Sharma's maps might be constructed as a kind of graph involving paths from architecture options to objective spaces. The conclusion is that the primary artifact used to express elements and relationships is some kind of graph. The DSL of Experiment 3 also bounds a category of graphs which lie within the DSL. Of course, what is described in the DSL is not a normal product architecture, but the architecture of a SE method.

The graphs of the SE methods have several archetypes. Figure 10.2 illustrates the potential characteristics of the method model permissible by the DSL in Experiment 3. Figure 10.2a is representative of a directed graph which proceeds "linearly," meaning in simple terms that there is neither branching nor iteration. The path through this sort of graph archetype is analogous to SRD in the OOSEM-lite of Friedenthal and Oster, the primary difference being that SRD has five nodes instead of three. Figure 10.2b illustrates a directed graph where there is a branching point. There are two



(a) Directed graph in linear form

(b) Directed Graph with Branching Point

(c) Directed Graph with Cycle/Iteration

Figure 10.2: Graph Archetypes in SE Methods

potential paths through this directed graph. This can be understand as alternatives, where either one task will be completed or another, like if-then-else. The use of the `ConditionalAction` might generate this kind of branching in some use cases. However, in the use case demonstrated by SAMD, Figure 10.2c is representative — in this directed graph, there is a cycle present which may cause iterative paths. This is highly representative of SA in general and SAMD in particular. In fact, the greater SA process with goal development from Experiment 2 illustrates clearly these sorts of cycles.

In this discussion already there are two boolean parameters which can be considered. The first is whether the method has branching characteristics. In the DSL, branching technically includes both Figure 10.2b and Figure 10.2c, it is just that the graphs have different characteristics depending on the kind of graph (Figure 10.2b is a Directed Acyclic Graph, but Figure 10.2c has a cycle). To capture this difference, the second boolean parameter indicates whether the method has iteration. If the method has iteration, then it is also branching in nature — both require `ConditionalAction` in the DSL.

In summary, the assertion here is to establish first two parameters which help to define the nature of the method model.

Table 10.1: Variables describing the methods according to graph archetype

Variable	SRD	SAMD
Condition	0	1
Iterative	0	1

Table 10.1 gives the settings for these variables for SRD and SAMD. Of particular note is how to discuss the iterative nature of SAMD, building on Experiment 2.

Criterion or Constraint for Iteration

One aspect of iterative behavior in the DSL and in particular in SAMD that is interesting is non-determinism. That is, while the number of state variables added or removed from consideration is a random variable, so too is the number of iterations. This is fitting for a method where the activity must continue as long as any state variables remain which have not been dealt with. However, perhaps it is not clear thus far how the DSL has in fact parameterized this behavior. To consider further what is going on in SAMD iterations, first principles will be used to construct a mathematical model of the process.

SAMD has an initial value of state variables, let it be called q. Each `ProcessNode` which modifies the state variables does so with a uniform random distribution. As a numerical approach will be applied here consider these distributions as discrete, with probability mass functions (PMF) $U_i\,(a, b)$ defining the distribution on the bounds $[a, b]$, e.g. $[0, 4]$ or $[-10, -2]$ from Experiments 2 and 3. A numerical approach will be applied here for simplicity — while rescaling an Irwin-Hall (IH) distribution may be more efficient from an analytical perspective, the approach here will aim for clarity of understanding. That is, to add the PMFs assigned to the `ProcessNode`s, the convolution will be used directly. Considering that in the case where there is no iterative step in SAMD, the distribution would have been some sort of IH_3, but if the iteration proceeded just once, then a convolution would still be necessary as $IH_3 + IH_4$. Con-

sidering the convolutions more directly permits to quickly, through this application of probability, test the rearrangement of the placement of the `ConditionalAction` — parameterizing in terms of the output. Regarding the output, the objective here is to obtain the probability that the number of state variables is greater than zero at the end of the iteration. Effectively, the result is a form of cumulative distribution function, where it may be see how likely particular conditions of the SAMD method proposal are to result in many iterations of model development.

The approach is as follows:

1. Specify the initial number of state variables q
2. Specify $1..n$ distributions by bounds $[a, b]$
3. For each distribution, generate the PMF U_i
4. Establish a tolerance as a stopping condition
5. Specify the node grouping — will $(3, 4)$ be the groupings as in the default case, or will the nodes used in the calculation differ from the default — representing an alternative method?
6. Calculate the probability per iteration

Source code for this implementation is included in Appendix A Section A.3. The PMF U_i is defined as as (x_i, p_i) for x_i from $[a, b]$ and p_i the probability according to the discrete uniform distribution.

The stopping condition is according to whether

$$abs\left(P(num_{sv} > 0)_i - P(num_{sv} > 0)_{i-1}\right) < tol$$

for tolerance value tol, and P the probability of the distribution for the condition $num_{sv} > 0$ number of state variables greater than zero.

The node grouping relation is defined as:

$$H_j = \overset{n}{\underset{i=1}{\circledast}} U_i = U_1 \circledast U_2 \circledast \cdots U_n$$

For a node grouping $(3, 4)$, $j \in [1, 2]$ with $H_1 = U_1 \circledast U_2 \circledast U_3$ the addition of PMFs by convolution, and $H_2 = U_1 \circledast U_2 \circledast U_3 \circledast U_4$. Both H_1 and H_2 are pre-computed and saved for later usage in the probability per iteration.

Probability per iteration is defined according to the relation:

$$p_{g0}(k) = \begin{cases} \sum_{x=0}^{x_{max}} A(H_1, q)(x) & \text{if } k = 0 \\ \sum_{x=0}^{x_{max}} A(D_{k-1}, k * H_2)(x) & \text{if } k > 0 \end{cases}$$

The function $A()$ represents either scalar addition or PMF addition by convolution. In the first iteration, the PMF must be offset by the input initial number of state variables q. Subsequent k iterations are modified by adding H_2 k times to the previous distribution D_{k-1} — the previous result of $A()$. If H_2 is only added once for each iteration to D_{k-1}, the convergence criterion will usually fail for any input conditions, and therefore this appears to the necessary mathematical structure of the problem.

To make the implicit parameterization explicit, 7 cases are defined and detailed in Table 10.2. The resulting probability per iteration from running these cases is shown in Figure 10.3.

Clearly, there are many variables which might affect the resulting distribution. Not captured in these cases is the possibility of adding a fifth distribution or more representing some other action applying `uniformAddInt` from the DSL. However, the groupings variables are an abstraction that represent changing the usage of `nextproc` and the dependencies from one `ProcessNode` to another. By changing the groupings, the network of tasks used in the probabilistic model is updated. These variables even permit excluding some nodes.

Table 10.2: SAMD Interation Parameterization

Variable	Values
Initial SV	10
Tolerance	1×10^{-5}
Default Case	
PMFs	$U(0,2)$, $U(-10,-2)$, $U(0,4)$, $U(0,4)$
Grouping	$(3,4)$
Case A (Slower Default)	
PMFs	$U(0,2)$, $U(-5,-2)$, $U(0,4)$, $U(0,4)$
Grouping	$(3,4)$
Case B (Faster Default)	
PMFs	$U(0,2)$, $U(-10,-5)$, $U(0,4)$, $U(0,4)$
Grouping	$(3,4)$
Case C	
PMFs	$U(0,2)$, $U(-10,-2)$, $U(0,4)$, $U(0,4)$
Grouping	$(2,4)$
Case D	
PMFs	$U(0,2)$, $U(-10,-2)$, $U(0,4)$, $U(0,4)$
Grouping	$(1,4)$
Case E	
PMFs	$U(0,2)$, $U(-10,-2)$, $U(0,4)$, $U(0,4)$
Grouping	$(1,3)$
Case F	
PMFs	$U(0,2)$, $U(-10,-2)$, $U(0,4)$, $U(0,4)$
Grouping	$(1,2)$

Figure 10.4 illustrates all the options run in the cases for the grouping variables. Specifically, Figure 10.4a is representative of the default case as well as cases A and B, and the SAMD description in Experiment 2 and 3. In this description, there were four nodes arranged such that in the 0th iteration, three nodes contributed to the state variable count, and in subsequent iterations, four nodes contributed. The other networks show the variability implied in the parameterization of the groupings. Figure 10.4b would change the placement of the `ConditionalAction` so that if the condition succeeds it proceeds to some other node, which then directs to the starting node (the first node which alters state variables). This would represent a fundamental change in the task specification of SAMD present in case C, and thus case C is no

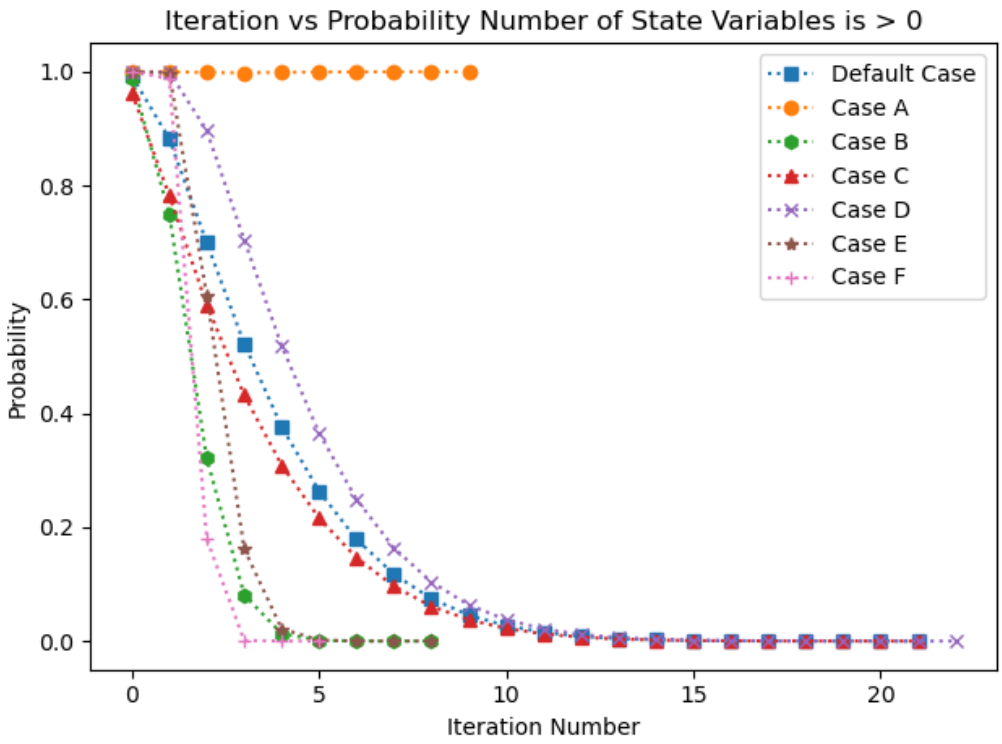
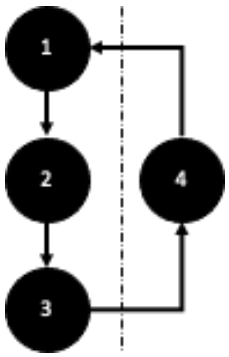


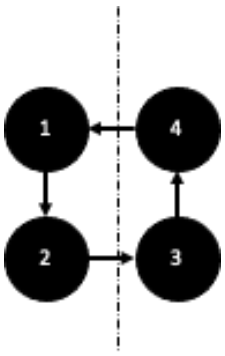
Figure 10.3: Probability that State Variables will be greater than zero for each iteration across multiple cases

longer representative of the method described by the source material. Further changes are possible to simulation in this vein, including cases D (Figure 10.4c), E (Figure 10.4d), and F (Figure 10.4e). As the second value decreases, nodes are excluded from the model. For this reason, fewer distributions cause the number of state variables to increase, and Case F results in a rapid completion of SAMD according to Figure 10.3. However, case B shows that changing the node structure is not the only route to accelerating SAMD. Without altering the task plan, case B modifies the number of state variable models which are established, reducing the pool of unhandled state variables faster. The apparent acceleration of case B in Figure 10.3 is because the $U(-10, -5)$ distribution ensures that many more state variables are handled per iteration than $U(-10, -2)$ for $q = 10$.

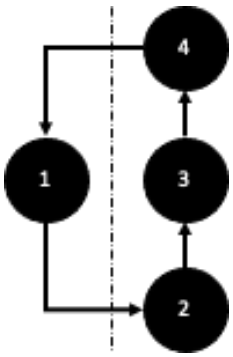
Regarding how the discussion on SAMD might apply to SRD, note that the struc-



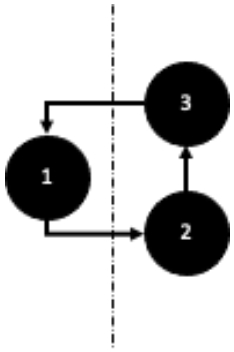
(a) $(3, 4)$ Grouping



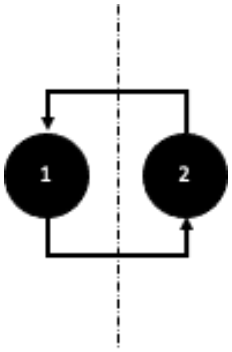
(b) $(2, 4)$ Grouping



(c) $(1, 4)$ Grouping



(d) $(1, 3)$ Grouping



(e) $(1, 2)$ Grouping

Figure 10.4: Groupings of nodes

ture of the graph for the SRD model is much simpler per Figure 10.2. As such, any mathematical model or simulation of the sort discussed above would likely serve just to get the the distribution obtained by Experiment 2 for the estimated duration of the method proposal by another means, at least, to obtain the PMF of this distribution rather than to sample the value via Monte Carlo simulation. There is always more than one way to view and exercise a mathematical model as a simulation; here, especially for SAMD, approaching the method model from the analytical and numerical perspective gives much more insight into the affects of parameters on the outcome. For SRD, it is safe to expect that the results seen by the mathematical approach should mirror the DES results, and no further information is expected. However, as the probabilistic method might be more straightforward to implement, it may be better applied in a management setting; that said, the DES technique followed from

the Business Process literature as in Jahangirian et al[46] as a primary candidate for these sorts of method models, as SE must necessarily be a Business Process.

Already, the results above for the DSL and specifically for SAMD indicate that insofar as number of iterations may serve as a proxy for time, as well as directly via SRD durations, therefore also by extension cost, that already the parameterization is capable of delivering insights on the conditions which must be achieved to succeed in using a method — and providing insight towards Hypothesis 1.

Summary of Parameters

Table 10.3 gives an overview of the parameters discussed thus far. These variables are generalized to cover both SAMD and SRD according to the DSL.

Table 10.3: Summary of All Variables for DSL

Variable	Values
Condition	0 or 1
Iterative	0 or 1
Initial Value of Passthrough	10
Tolerance (as required)	1×10^{-5}
Task PMFs	$U_1(a_1, b_1) \cdots U_n(a_n, b_n)$
Groupings	$(p_1, q_1) \cdots (p_m, q_m)$

All of these parameters might be established either explicitly or implicitly by parsing the DSL in the SysML model from Experiment 3, except tolerance. Most clear is the representation of the distribution bounds in the `ProcessNode` data, and since for SAMD these quantities had nearly as much effect as changing the structure of tasks and thus the method itself, the power of these variables must not be underestimated. Further, note that the difficulties in changing variable spaces discussed especially by Kerzhner and Sharma through architectural selections come into play as nodes are added or removed in the PMF list and/or in the applied Grouping. That is to say that the addition of another `ProcessNode` (which affects the statistical representation, thus including a `unformAddInt` action in the DSL) has the impact of adding several

variables to the model, at a minimum two new parameters for the distribution. To be clear, `ProcessActions` which do not affect the passthrough value are not considered in these simplified representations for the sake of parameterization, as they do not affect the passthrough value (e.g. printing the description of the task). Changes to the `nextproc` actions may show up as changes in Groupings here, or in the set of available PMFs. Additional binary variables may be required to fully parameterize the simulation architecture; however such a parameterization may not be necessary for the goal of the end user to evaluate a method proposal.

10.2.1 Techniques to Specify Simulation Cases and DoE

In attempting to specify cases for simulation, first consider the likely usage of the tools presented here and in Experiments 2 and 3. Notably, none of these simulation tools provide any information as to what work must be done for the SE process; this knowledge comes from the systems engineers and subject matter experts. Thus, it is safe to assume that the underlying network of the method model is unlikely to be the subject of variation during these simulations. Rather, the objective for these simulations is likely to be to determine necessary process measures which ought to be tracked to determine if the work is proceeding according to plan. For example, whether the rate of state variables addressed during SAMD iterations is within expectations. Perhaps some Walworth-style variable modifications (aka greater "intensity") may be required if the number of state variables addressed per iteration falls below expectations; however, what "intensity" or similar variables actually mean is highly dependent on specific definitions and interpretations. This is especially true if considering how to measure data with which to calibrate a model of this sort.

Therefore, in considering running several cases or even a Design of Experiments (DoE), the main target is likely to be the distribution variables. For example, in SAMD, the reason would be to explore the sensitivity in number of iterations (perhaps

for 95% certainty of finishing SAMD given q) for a given description of the method proposal. For SRD, the corresponding distribution parameters could be similarly varied to describe the sensitivity of the durations. To do this, consider that $a < b$ for the uniform distributions used here. Thus, the parameters to vary are 1) either a or b and 2) the difference between them with $b - a = diff > 0$. For this purpose, SRD would have 10 variables while SAMD would have 8. If including q, then the number of variables would be 11 and 9, respectively. The DoE for this quantity of variables could then be formulated to the modeler's preferences.

The next technical hurdle is how to apply the cases of the DoE to the data in the DSL. Of course, running these cases in a simplified manner as prescribed above for SAMD might reduce the overall technical complexity. However, if managing the DoE runs via the DSL, the main issue for implementation comes down to time and how many cases must be run. For example, if very few cases will be run, then manually modifying the `use` dependencies to point to data instances with the appropriate DoE values is practical, if tedious. If there are many cases, then some other approach will be necessary. At again a simplistic level, the entire proposal in the SysML model could be copied and pasted n times, modifying the data instances in the copies. Then, each of the n copies could be run, and in fact it is possible to construct an analysis which runs n analyses that use the n copies of the method proposal as the input for the simulation. Other solution procedures would require changes to the external software, or to the DSL and internal scripts within the system modeling environment.

10.2.2 Additional Criteria for SE Methods

Importantly, as a business and moreover social process, there are qualitative and subjective parameters which should be considered in comparing SE method proposals. For example, an organization more engaged in exploiting their market and technology that developing new technology may be interested in SRD, which presupposes

that all models etc are already available. Meanwhile, a different organization more engaged in exploring new technology may require a method which explicitly calls for the construction of new models; such an organization may find SAMD a better fit. While these distinctions may appear soft or subjective in the engineering domain, they are the subject of serious study in the business domain with research techniques such as sentiment analysis and search corpora of articles for specific terminology[125]. Specifically, Uotila et al[125] use the terminology put forward by March[126]. March describes exploration as characterized by "search, variation, risk taking, experimentation, play, flexibility, discovery, innovation" while describing exploitation by "refinement, choice, production, efficiency, selection, implementation, execution"([126] as described by [125]). While both authors discuss the need to balance these approaches, and Uotila et al even try to show whether some enterprises practice an optimal balance, the key takeaway is that the organizational character is an important factor in the selection of the method proposal and/or tailoring thereof (e.g. as described by [6] for tailoring of the SE process more generally).

Furthermore, there is also an aspect of model-orientation. Practitioners of SAMD come from a mindset of autonomous systems, for which models must come first and precede much of the development in order to specify and perhaps provide some guarantee of performance, especially for critical systems like spacecraft which may have a single opportunity to achieve an objective. For these practitioners or for those who have only known SE as MBSE, SAMD may be quite natural. However, practitioners of more traditional SE methodologies such as described by [11] may appreciate the resemblance of SRD to activities described in older, traditional SE methodology, even if the prescribed tools and environments and even the specific tasks are very different than in decades past. These concerns also speak to the character of the organization, but instead of the business strategy, they appeal more towards the personnel and their conception of the product development technique. A methodology may be preferred

if most of the engineering staff are cognitively primed to accept it, thus increasing the likelihood of compliance and therefore results in ensuring that the system is developed and deployed successfully. Consider that in solving a new problem, developing a new system, or meeting new customer needs, that the cognitive load of solving this problem is of prime importance, and that additional cognitive load due to following new and perhaps arcane methodology may be unwise. In fact, in the MBSE literature, this may be the driving motivation behind the development of DSLs to aid in the adoption and application of systems modeling languages for particular purposes and experts.

Related is the extent to which a SE methodology is generalizable to many domains, many kinds of vehicles, and many systems of interests. Software and hardware frequently have different product development methodologies, and often the current trends in software engineering are appealing for hardware engineers to attempt to adopt — see e.g. Agile. However, more practically, consider that while controls engineers or autonomy specialists may appreciate that many of their terms and constructs are directly acknowledged by SA, a structural engineer or widget designer may feel left behind — or worse, that the methodology does not apply. Clearly, SRD and OOSEM-lite have less domain flavor – they are more domain agnostic — and thus may stand to gain broader appeal among the wide variety of engineers and specialists involved in vehicle design and development, especially in domains where autonomy is less important.

Finally, neither of these methods are particularly novel, and in general SE has existed for some time. Presently, most concerns are regarding the adoption, use, and exploitation of new tools and environments for accomplishing SE tasks in the domain of MBSE. In this case, however, both SRD and SAMD have a lineage over 15-20 years and are MBSE methodologies. As a multi-decadal transition, perhaps beginning with Wymore[16] in 1993, MBSE itself is rapidly maturing in various communities

of practice, especially as these communities refine the tools and environments (e.g. languages and language editors) for their purposes. Given this history, there is a possibility that organizations have already deployed methods related to SRD and SAMD. In this case, consideration of tweaks to existing methods or adoption of new methods may be informed in part by past usage of methods and preferences derived from such experience.

10.3 Exploring the Models

In exploring the models, it is appropriate first to summarize what has already been done. In Experiment 1, the Walworth model was run on a DoE and some of the data from that set of cases was plotted. However, due to the outputs being time series, summarizing and simplifying the output of the cases is more difficult. That said, Figure 7.2 provides the kind of view which may be possible for the outputs across all cases: a 3D plot where one axis is time, one is the magnitude of the response (e.g. "Work to be done"), and the third is the case number. For the DES models in Experiment 2, model exploration was performed by Monte Carlo simulation instead of DoE. The Monte Carlo simulation enables sampling of an output distribution from a model based on several random variables. Experiment 3 enabled running individual cases from a system model on the analyses from Experiments 1 and 2. Finally, above, additional techniques to explore the parameterization and impact of parameters on the outcomes of, for example, the SAMD method proposal, can be explored in a way that is compatible with the content of the system model from Experiment 3. However, perhaps the most meaningful demonstration of exploring a set of models would be to apply them to a new problem. In this case, a new method will be simulated. Finally, relations to cost will be discussed.

10.3.1 Demonstrating Variability and Exploring the Model Space by Construction

Ahead of Experiment 5, and as a candidate validation procedure, the RDS process will be formulated with the DSL for simulation to demonstrate variability and to explore the model space through construction. These actions constitute a manual manipulation of aspects of the modeling process which would need to be formulated as some sort of discrete variable per the literature reviewed earlier for automated exploration. RDS was previously introduced according to Mavris et al[71]. To reiterate, the overarching purpose is to enable rapid exploration of the system design space and discovering design solutions which are robust to perturbations in the solution environment, according to a constructed form of probability of constraint success. As a result, the output of the process includes a design vector, target/constraint values if updated from initial or baseline values, as well as a set of surrogate models, margins, and spaces of candidate feasible design points. The technique is analysis-oriented or model-driven in the sense that models for the design of systems are central to the discussion. Specifically, the models ought to perform some form of sizing, that is, determining a mass or weight or volume, subject to constraints, and for this sizing calculating performance for subsystems. A specific technique of interest for accelerating the use of concepts like Monte Carlo simulation in this process includes surrogate modeling, specifically of the subsystem performance relations, which helps improve the computational feasibility of the technique. The usual role of this technique is to help bound the space on which requirements for the product, its technologies, and its environment may exist given the current understanding and assumptions, and thus it is considered here.

The overall steps for RDS are portrayed in Figure 10.5. The `ProcessNodes` are displayed side-by-side with the RDS methodology illustration from Mavris et al[71]. This illustrates which nodes correspond to which steps in the paper's method, and also how they are subdivided. Some items are subdivided, such as running cases vs.

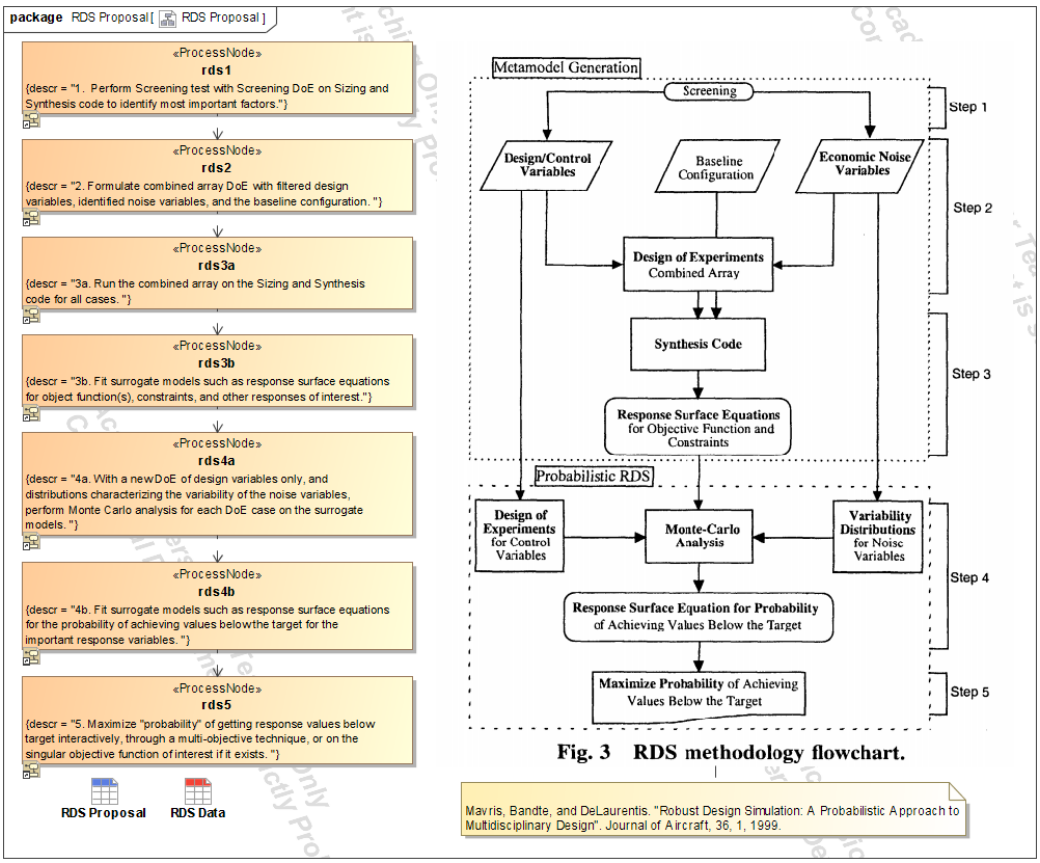


Figure 10.5: RDS Method proposal formulation in the DSL

fitting models. The reason for this is that running cases often takes a substantial amount of time. In the source material, even the DoE and Monte Carlo simulation of the surrogate models took over 4 hours. The subsequent fitting activity relies heavily on human involvement to judge the goodness of fit for the response surfaces, or other surrogate models. The interactive activity of fitting these models may take some time as well, and is therefore tracked as a separate `ProcessNode` in 3b and 4b.

#	△ Name	○ passval : Integer	○ minval : Integer	○ maxval : Integer	Process Node Usage
1	d-rds1	0	1	5	rds1 rds4a
2	d-rds2	0	1	3	rds2 rds3b rds4b
3	d-rds3	0	3	15	rds3a
4	d-rds4	0	1	4	rds5

Figure 10.6: Data used by RDS Method proposal

Figure 10.5 does not display the `use` relations. These are hidden and portrayed instead on a table via the available data, seen in Figure 10.6. Nodes 1 and 4a focus on running DoEs either on the actual sizing and synthesis code with fewer cases for screening purposes, or running many cases for Monte Carlo simulation over surrogate models, and are represented here for simplicity by the same distribution of time $U(1,5)$. Steps 2, 3b, and 4b are characterizing by a mix of automated and manual processes, and require a person to interact with some product (DoE, results, fit metrics) and make decisions about what to use. Therefore, they are each assigned $U(1,3)$. The most expensive step is 3a, in which the combined array DoE is run on the full sizing and synthesis code, and this is given $U(3,15)$. The reason for the larger range on 3a is to cover the possibility for running more or less expensive models as part of the analysis, or for re-running failed cases, running duplicate cases for controls, or other contingency. Finally, step 5 is assigned $U(1,5)$, as this step may involve an automated procedure, or it may involve an interactive session, where decision-makers can adjust targets to set requirements on the system and its technology.

#	Name	descr	Next ID	Process Action List	Data	General Node
1	rds1	1. Perform Screening test with Screening DoE on Sizing and Synthesis code to identify most important factors.	rds2	44 print 45 uniformAddInt 46 nextproc	{"maxval":5,"minval":1,"passval":0}	Task
2	rds2	2. Formulate combined array DoE with filtered design variables, identified noise variables, and the baseline configuration.	rds3a	47 print 48 uniformAddInt 49 nextproc	{"maxval":3,"minval":1,"passval":0}	Task
3	rds3a	3a. Run the combined array on the Sizing and Synthesis code for all cases.	rds3b	50 print 51 uniformAddInt 52 nextproc	{"maxval":15,"minval":3,"passval":0}	Task
4	rds3b	3b. Fit surrogate models such as response surface equations for object function(s), constraints, and other responses of interest.	rds4a	53 print 54 uniformAddInt 55 nextproc	{"maxval":3,"minval":1,"passval":0}	Task
5	rds4a	4a. With a new DoE of design variables only, and distributions characterizing the variability of the noise variables, perform Monte Carlo analysis for each DoE case on the surrogate models.	rds4b	56 print 57 uniformAddInt 58 nextproc	{"maxval":5,"minval":1,"passval":0}	Task
6	rds4b	4b. Fit surrogate models such as response surface equations for the probability of achieving values below the target for the important response variables.	rds5	59 print 60 uniformAddInt 61 nextproc	{"maxval":3,"minval":1,"passval":0}	Task
7	rds5	5. Maximize "probability" of getting response values below target interactively, through a multi-objective technique, or on the singular objective function of interest if it exists.	null	62 print 63 uniformAddInt 64 yieldtime	{"maxval":4,"minval":1,"passval":0}	Task

Figure 10.7: RDS Method proposal in tabular format

The entire method proposal is summarized in table format in Figure 10.6. Each `ProcessNode` is very similar to the `ProcessNode`s used in SRD. In fact, the first four from SRD are equivalent to the first six in RDS, and the last SRD node is the same as the last RDS node. However, while the archetype of the network is the same (Figure

10.2a), there are additional nodes in RDS and the precise distributions are different. In this way, the space of candidate methods described by the DSL can be explored, at least by exploiting the aspects which are easily modified.

If satisfied with the method proposal, analysis of the method proposal proceeds as the next step. Figure 10.8 illustrates the formulation of the analysis. In this

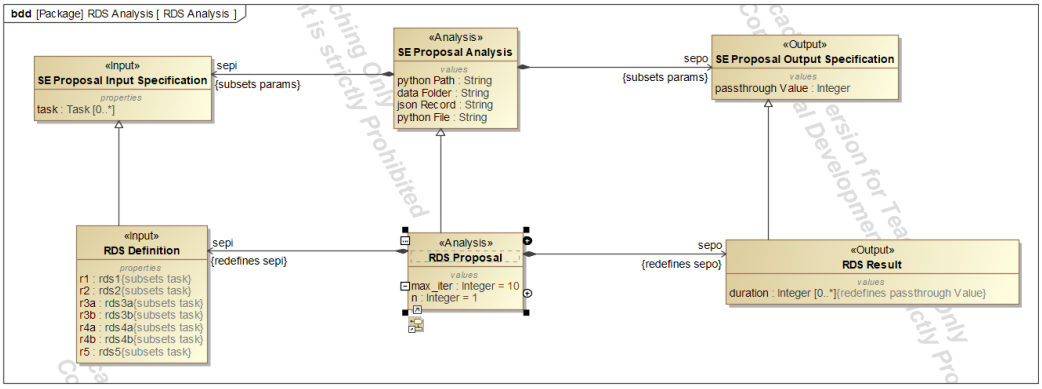


Figure 10.8: RDS Method proposal analysis formulation

case, unlike in Experiment 4, the output data will be captured, showing variability also of the analysis formulation. The output data in this case includes multiple result quantities of the duration value from the passthrough variable. Figure 10.9

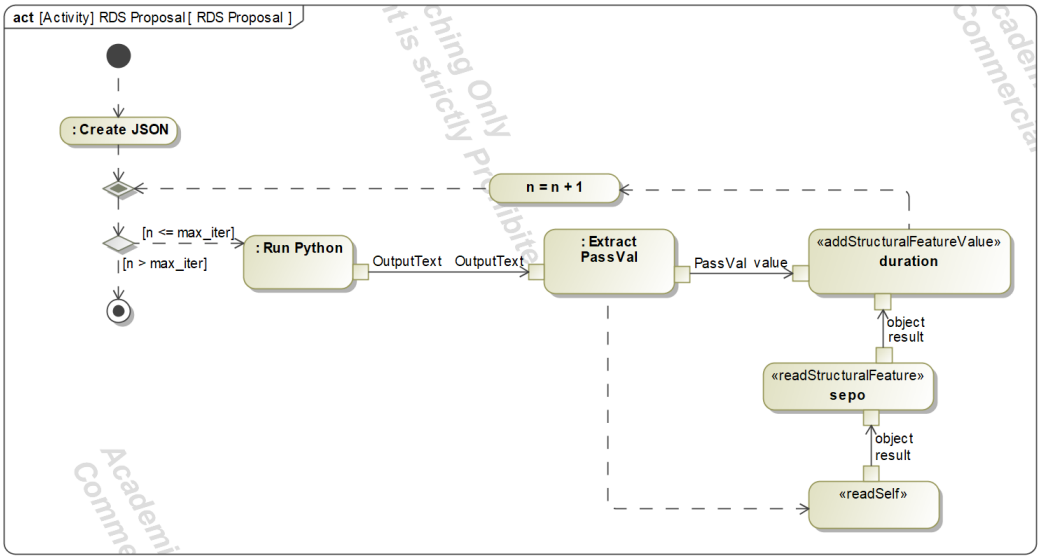


Figure 10.9: RDS Method proposal analysis formulation

illustrates the modified analytical procedure by which the analysis runs the code

from Experiment 2. In this case, most of the scripts in the system model are identical from before, but they are subdivided different to permit for iteration. The analysis will iterate 10 times. Each time, it will run the Python code, retrieve the output, and then, unlike before, pass the output to another script for parsing before the resulting integer value is saved to the system model. The data resulting from running the analysis is seen in Figure 10.10. In Figure 10.10, there are fewer than 10 values.

#	Name	duration : Integer
1	rds proposal.sepo	22
		25
		14
		19
		17
		16
		21

Figure 10.10: RDS Method proposal analysis formulation

Some values are lost when the simulation is automated. The likely reason for this is issues stemming from how the Python process is launched from the script, as well as potential for the fUML actions to improperly append values by popping some out of the list, as unlike the record in Figure 10.10, all of the cases are properly written to console. Running the analysis manually permits capturing more of the values, but requires extra steps to save the data. This issue is a limitation perhaps of the environment applied in this example.

The result here clearly shows that the techniques established permit exploring the different possibilities of SE methodologies — if not by exploiting a large number of discrete variables, at least by construction of the method proposal. Clearly, the combined environment enables proposing various SE methods, simulating them, and inspecting whether they are compatible with the product development plan or timeline for launch. Furthermore, by building up probabilistic forecasts of method

performance, an organization can therefore have a better idea of how work is progressing, *enabling SE modeling and planning.*

10.3.2 Consideration of Cost

A key aspect of Hypothesis 1 is the supposed correlation to cost and schedule from leading indicators. Of course, the Walworth model for requirements status was not a very great success as far as the information gleaned about the methods, though it was successfully implemented. Considering instead the direct simulations of the Method proposal, cost is still relevant and perhaps more directly so. As discussed in Experiment 2, it may be possible to assign per unit of time or per iteration of model development a range of cost according to organizational experience. This relation would allow transforming the result of the passthrough variable directly to cost and obtaining distributions of cost. However, the uniform distributions applied in the Python tool and the DSL are perhaps less appropriate for direct cost modeling than for performing a generic probabilistic assessment. Additional capability for distributions should be added for direct cost modeling, as well as perhaps multivariate passthrough between `ProcessNode`s.

10.4 Direct Comparison of SE Methods

Table 10.4 presents all SE Method proposals discussed thus far alongside the criteria discussed throughout Experiment 4. The subjective or qualitative criteria discussed earlier are somewhat abbreviated as Explore/Exploit, Model Orientation, Generality, and Experience Adaptable. Explore/Exploit follows directly from the earlier discussion in two categories. Model Orientation is list as incidental or first. For SRD, a model is built incidentally to the tasks at hand, as typical SE tasks are performed as one might be familiar from tradition, but they are done within a system model, constructing what is primarily a semantic model. For SAMD and RDS, models in a

mathematical sense come first. Generality is the degree to which the method proposal is generalizable. SRD is derived from OOSEM which can be adapted to any vehicle or industry which traditional SE methodology might be applied (submarines, aircraft, spacecraft, etc). SAMD is slightly less generalizable. RDS is highly generalizable, and the techniques underlying it have been applied to many domains in aerospace engineering, mechanical engineering[127], and elsewhere. Experience adaptable is a statement about the kinds of experience to which the method relates, whether traditional systems engineers, controls and autonomy engineers, or disciplinarians/analysts. The rest of the variables are the variables from the method proposals themselves, listed similar to before, but for each methodology considered throughout Experiment 4.

10.5 Conclusion

Experiment 4 aimed to prove Hypothesis 1 as to whether the over all platform for SE modeling and planning could enable decision-making on the SE method proposals for SE processes according to measures, such as leading indicators, which may be correlated with lagging indicators such as cost and schedule. In this case, While the targeted leading indicator model can be exercised, it does not produce much insight. However, the method models generate substantial information, characterize method performance, provide probabilistic assessment, enable setting targets and process requirements, and provide the capacity to study variability in method specification both in terms of the limits applied to variability as well as the task network itself. Furthermore, unlike traditional forms of analysis, as discussed in Experiment 2, these task networks can include cycles. After a discussion on architecture exploration, a mathematical model was used to demonstrate that the underlying theory of Experiment 2 and 3 performed this kind of exploration, and to show that by changing the method specification content, vastly different performance would result. These changes in performance were shown to be realized both by tightening performance on

Table 10.4: Comparison of criteria for each method proposal

Variable or Criterion	SRD	SAMD	RDS
Explore/Exploit	Exploit	Explore	Explore
Model Orientation	Incidental	First	First
Generality	High	Moderate	High[127]
Experience Adaptable	Traditional SE	Autonomy and Controls	Disciplinary Analysis
Conditional	0	1	0
Iterative	0	1	0
Tolerance	N/A	1×10^{-5}	N/A
Groupings	N/A	$(3, 4)$	N/A
Task PMFs	$U(5, 10)$ $U(5, 10)$ $U(5, 10)$ $U(5, 10)$ $U(5, 10)$	$U(0, 2)$ $U(-10, -2)$ $U(0, 4)$ $U(0, 4)$	$U(1, 5)$ $U(1, 3)$ $U(3, 15)$ $U(1, 3)$ $U(1, 5)$ $U(1, 3)$ $U(1, 4)$

the existing network, as well as changing the task network itself. Furthermore, additional criteria of subjective or qualitative nature were acknowledged and established to provide a broader picture of the method proposal consideration. Then, to illustrate constructively how the DSL of Experiment 3 enables method proposal exploration, a third method RDS was represented and simulated from the system model. After a brief discussion on how the method proposal simulations relate to cost per Experiment 2, the overall picture of the methods and criteria or variables was given by Table 10.4. Systems Engineers and project planners can look at this table, apply their preferences, and perhaps make some decisions or even revisit the simulation in the system model to compute new performance expectations, all while remaining within the system model and incorporating SE planning in MBSE. Altogether, Hypothesis 1 is accepted as the platform established across Experiments 1, 2, 3, and 4 enables the exploration of SE methods via SE task performance prediction, and brings all the information together in the system model for modeling and planning purposes.

11

EXPERIMENT 5: IMPROVEMENTS FOR MDO AND RDS REPRESENTATION IN MBSE

Tight integration of modeling and simulation to the MBSE language helps to support system validation as well as model validation. These actions are especially important in the early phases of the life cycle. Ensuring that the model representation embedded within an analysis behaves as expected is an important part of this integration and answering Hypothesis 4, and the formal techniques by which this can be done have been established for many years. Implementation-wise, it will be similar to generating the simulation of the SE process from its description in SysML; however, the analytical model produced will be for a design problem. These sorts of transformations have been discussed in detail in the preceding chapters and continue to be a subject of research (e.g. [95]).

Experiment 5 seeks to answer:

Research Question 5 How should a DSL for RDS activities interface to M&S, uncertainty, and other infrastructure from a system model?

Research Question 6 Given a description language for an aerospace vehicle, how should a language for design optimization be interfaced to the description with support for probabilistic analysis?

Experiment 5 seeks to determine:

Hypothesis 4 If a labeling for probabilistic methods is applied to a customization of an MBSE language with transformation via intermediary representation to target analytical environments, then confidence will be increased in the system representation of the resulting analytical models.

For the scope of this experiment, it is assumed that the mission requirements and mission or orbital design have been successfully concluded per Friedenthal and Oster[98]. The focus will be placed on the "Specify System Requirements" activity in the canonical SE methodology. Key data that must be taken as given includes the number of spacecraft, the geometry of the orbits, launch considerations, and mission life analysis. Thus, the interest here is for the design of a spacecraft given a particular mission. Below is the information previously covered about Experiment 5.

This technique has been used twice before by the author for incorporating MDO with MBSE languages, first in 2017 for a space vehicle problem and again in 2019 for an aircraft problem. The following must be constructed:

$$\Gamma_{S_v} \underset{L}{\mapsto} \Gamma_{I_v}$$

$$\Gamma_{I_v} \underset{M}{\longmapsto} \Gamma_{E_v}$$

In this case, Γ_{S_v} is some subset of a SysML model of the vehicle with a particular labeling, Γ_{I_v} is an intermediate representation of the vehicle content, and Γ_{E_v} is the external representation of the vehicle. This is close to the approach advocated by Cole and Dinkel [64], however it does not require that all the content of Γ_{S_v} is translated to Γ_{I_v}, nor does it require that Γ_{I_v} is in serialized form. Usually, Γ_{I_v} is a sort of list or associative array containing the desired information in memory. This is a less-formal transformation, where customization happens both at the level of Γ_{S_v} being a near template of the information needed by Γ_{E_v}, as well as within the sets of rules L and M. The rules L and M describe the translation between the model representations; in effect they form the model-transformation. In past experience, Γ_{E_v} is in JSON format to be ingested by the analytical program. The analytical program configures its analytical models as far as it is capable of in terms of Γ_{E_v}. One option which may be useful in particular for the probabilistic aspects of RDS

may prove to be probabilistic programming languages, which have seen much recent development[96, 97]. These languages are reportedly in the declarative programming paradigm, aligning with techniques discussed in Contribution [61, Paper A].

11.1 Scope of the Analysis Described in the Baseline

While the SysML model established by Friedenthal and Oster for FireSat does not actually provide much in the way of computation, it does specify several analyses in coordination with the requirements activities. Therefore, it is important first to understand the overall scope of the analyses used in this baseline system model[115]. An N2 chart is given in Figure 11.1. For example, the orbit analysis described in Friedenthal and Oster[98] is performed in STK. Thus the results from this particular analysis are likely to be unidirectional, and the parameters are unlikely to be bidirectional without substantial effort — software development or purchase of such software — in wrapping the analytical environment. Similarly, many other relations are stated simply or refer to other analytical environments. Thus, a reasonable assumption is made about the causality to produce Figure 11.1, as in reading equations from right to left as in $y = f(x)$, common across programming languages, where x is the inputs and y is the outputs. The N2 chart is subdivided according to the activities of interest: mission requirements derivation, spacecraft requirements derivation, and the detailed requirements derivation. Specifically, some subdivision on spacecraft systems is asserted by the authors, such that some subsystems of the spacecraft are determined in one phase and others in the subsequent, such as the payload (sensor), power system, and navigation, whereas propulsion is the primary focus of the vehicle or spacecraft requirements derivation. One subsystem or budget not included in the SysML model or textbook is the link analysis or link budget for FireSat as described by Wertz et al[91]. Thus there is a gap in the requirements analysis activities at the same level of detail as the analyses presented across Figure 11.1, and this gap will be

the focus of this chapter.

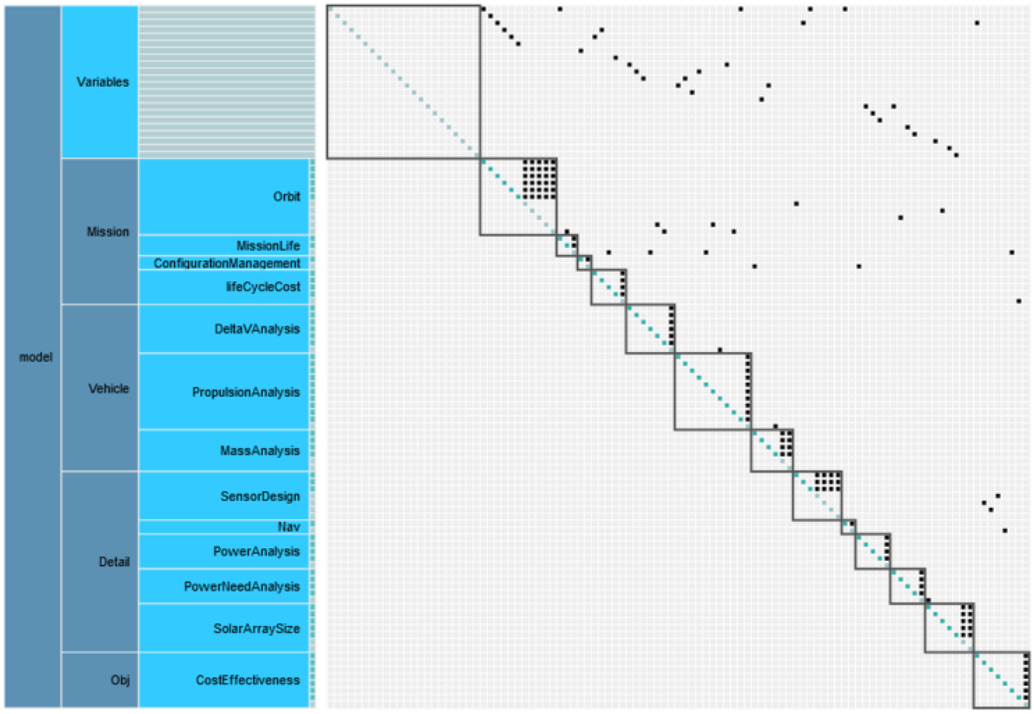


Figure 11.1: N2 Chart depicting the analyses, inputs, and outputs represented in Friedenthal and Oster[115]. This representation assumes that analyses flow in the direction implicit to the stated equations, though in principle under certain circumstances the causality of the variables may not be predetermined.

11.2 Analytical Technique illustrated for FireSat Design Space Exploration

As previously discussed, FireSat has seen significant attention in terms of Systems Engineering literature and analysis, including aspects of design space exploration.

11.2.1 Establishing the Vehicle Description

There are differing definitions of the level of detail which might be considered for the spacecraft design requirements, as well as what constitutes detailed component design[89, 90, 98, 91]. In the spacecraft specification which is established by Friedenthal

and Oster, they indicate that it covers: Launch Vehicle Mechanical and Electrical Interfaces, Fire Detection Probability and False Positive Likelihood, Spacecraft Attitude Pointing Accuracy, Spacecraft Data Downlink Capacity and Data Storage, Spacecraft Fault Management, Spacecraft δV, Spacecraft Mass, Power, Thermals, Payload Interface, Structural Integrity, Orbital Constraints, Reliability, Life, and Unit Cost[98, p. 66]. Paired with this requirements specification space, however, is a singular "Mass and Delta-V Analysis". Their accompanying model and website do not provide substantially more detail; much of the model is incomplete, though there are indications that the authors may have considered additional analytical capability. Additionally, the "Mass and Delta-V Analysis" is under-specified and cannot be used as-is. Also not completely justified here is how the power analysis is left for the step of "Synthesize Alternative Architectures" when the "Mass and Delta-V Analysis" includes attitude adjustments which assume an architecture using propulsive maneuvers instead of magnetorquers or other techniques. Further, by [98, p. 67], the text clearly establishes that the analysis only provides rationale for two (Mass and Spacecraft δV) requirements in the Spacecraft Requirements Specification. Notably absent also is navigation, a subsystem necessary for the mission performance criterion of location accuracy.

Several assertions can be made at this point:

1. Radiation dose over the life of the spacecraft is a key measure for faults
2. Solar and other irradiation drives thermal and potentially power considerations
3. Data transmission is critically important for observational payloads collecting imaging data
4. Navigation, Attitude Determination, and Attitude Control depend heavily on the mission environment and directly impact mission success through imaging, communication, power, and thermal concerns

While Friedenthal and Oster include some discussion of these topics at least tangentially, it is clear that much detail is left out. For example, Gross and Rudolph illustrate a relatively complete version of the FireSat analyses[89, 90] included substantial detail on subjects such as telecommunications/link analysis and radio design[128]. Due to lack of necessary detail in the established "Mass and Delta-V Analysis" alongside their limited scope, new analyses must be established. New analyses should include a link analysis for the telecommunciations capability in order to support the issues and objectives which are recognized in the baseline. Specifically, this implementation will compare the constraint graph style of [128] with a modified CBAM-style implementation as discussed in Contribution [61, Paper A] in order to address the literature and develop an analytical baseline, and dive in to look at issues directly relating to Hypothesis 4.

11.2.2 Understanding the Constraint Graph Approach

Gross and Rudolph[128] established that a constraint-graph approach, where the constraint graph is formulated in the SysML model and subsequently extracted for manipulation, can be useful for exploring trade studies as well as sensitivity studies through techniques for differentiation. In this experiment, the idea is to see how to better incorporate robust design simulation especially for studies in feasibility for validating design concepts and performing Systems Engineering validation activities.

11.2.3 The Constraint-Graph Approach May Enable Probabilistic Analysis

The constraint graph may play a role in determining a "robust" design solution, considering that the constraint-graph approach assisted Gross and Rudolph in exploring the FireSat design space. Taking this seriously, the first immediate concern ought to be how in fact a monte carlo simulation might be run on a constraint-graph representation. The reason for this is that the constraint graph takes the place of any or

all such "meta-models"[71] and that running multiple cases, for example as part of a Design of Experiments (DoE), is usually straightforward by changing the values of the causal inputs to the constraint graph model — though some limitations may apply in commercial software. However, for a robust design some probabilistic analysis is necessary in the trade study in order to determine the variability of responses according to priors on the input distributions. If it is asserted that the constraint-graph approach is sufficient for these studies, then there should be certain characteristics or criteria which arise in these models. Of note, these characteristics are software in nature, and may be more or less fulfilled by many different combinations of tools and environments for SE activities.

1. Ability to specify distributions for input or given quantities
2. Ability to automate simulation of many cases
3. Ability to analyze response data visually or otherwise
4. Ability to store data if necessary for future study
5. Ability to scale the approach to analyses of higher fidelity

In order to investigate these criteria, a constraint graph representation roughly equivalent to that in Gross and Rudolph's works will be constructed — effectively, repetition, yet also by extension filling the apparent gap from Friedenthal and Oster who assert the possibility of such techniques on their own system model.

11.2.4 Construction of a Constraint Graph

The first concern, according to the baseline studies, is the construction of a constraint graph representation in SysML related to a FireSat analysis. Note that the design compiler cited through the various works of Rudolph does not appear to be generally available, at least in English language literature, which brings together all

of the rules from works such as Gross 2016 [90] or Geyer 2008[85]. However, other capabilities are well-known such as described in Contribution [61, Paper A] and laid out in detail by Peak et al[129]. This technique specifically leverages the commercial software Paramagic which enables interpreting SysML Parametrics as a form of constraint graph which can be exported to a solver tool (OpenModelica, Mathematica, or MATLAB) for solution by symbolic algebra after selecting desired causality and providing necessary values. This capability includes trade studies, and with a larger commercial environment is capable of probabilistic analysis as demonstrated by Bajaj et al[130] — although, specifically, the monte carlo analysis was not done with Paramagic and may not have included direct representation of distributions in the SysML model. Note that the non-probabilistic portion of Bajaj et al focuses on FireSat. That said, this information aligns with the statements earlier that the concerns here for Hypothesis 4 are software in nature. Considering that this environment is overall promising, and may present some of the desired capability towards Hypothesis 4, the domain analysis will first be established as this kind of constraint graph as these tools (MagicDraw, Paramagic) happen to be available to the present author, as well as a small plugin for visualization of the constraint graph called Buzztoys Panorama[131], described in detail by Peak et al 2010[132]. Additionally, the presentation by Peak et al 2010 not only highlights the construction and visualization of these constraint graphs, but indicates that these environments argue for the mapping relation between system representation and analysis representation in SysML be parametrics — while the mapping relation outside the SysML editing environment is "native model relationship (via tool interface, stds[sic],...)"[132]. Here is another piece of evidence to note, which is important insofar as if such an interface is not readily available off-the-shelf, as it were, then the analyst and or systems engineer must be able to roll their own, and the assertion in this experiment is to do so via these intermediate representations[64] discussed earlier and in fact already applied in Experiment 3.

Returning to the subject at hand regarding the constraint graph. The task is to create a SysML model of the system of interest, and likewise a model of the analysis using SysML blocks which may provide some equations to calculate performance, in this case regarding antenna gain per Gross and link budget per SMAD 2003[133]. The representation in SysML using parametrics and specifically Paramagic is governed by a variety of patterns including the Context Based Analysis Model (CBAM) and Composable Object (COB) discussed in Peak et al 2007[129] among others which will be mentioned in passing. For example, a simple Analyzable Product Model (APM) is established for Gross's satellite link representation, providing some detail and a few relations regarding the satellite link object, transmitter, and possible antennas, as shown in Figure 11.2.

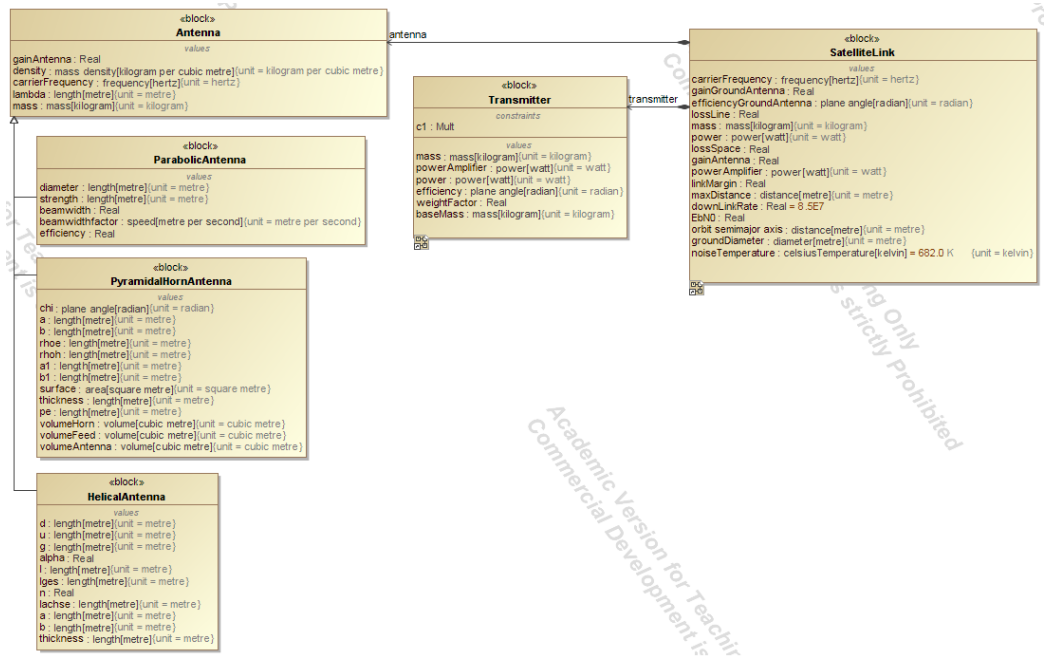


Figure 11.2: System Representation based on Gross and Rudolph 2014[128]

By constructing an analysis model which is specific to a particular design (e.g. a particular kind of satellite link), a CBAM is created. This CBAM is represented in this example by the Link Analysis block, which has as a part property the Satellite Link with its data available for use. The composite model is illustrated by Figure

11.3.

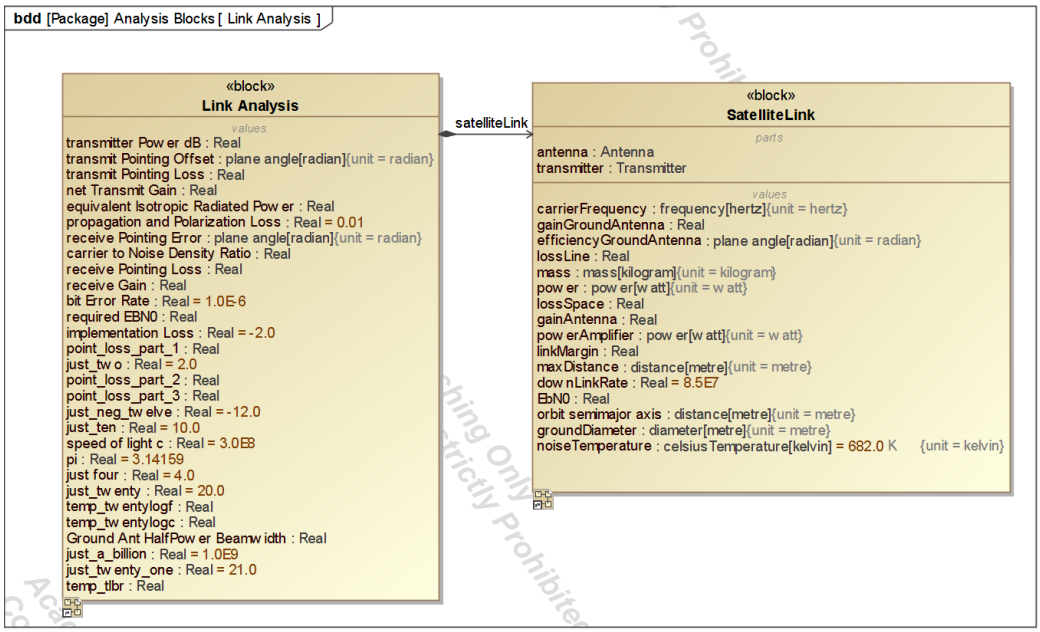


Figure 11.3: Link Analysis CBAM pattern

With the model structure established, including many useful quantities as value properties, the tedious effort of constructing the parametric diagram according to Paramagic style must follow, and the result is shown in Figure 11.4, which augments the block structures shown thus far with numerous Analysis Building Block (ABB) units regarding logarithm calculations constructed for re-use. Figure 11.4 is of paramount importance as the relations conveyed on it form the bulk of the edges, and most of the computations, required for the solution of the constraint graph outside relations in the APM.

Figures such as Figure 11.4 are not known for their readability and are required more for functionality in how a tool such as Paramagic parses a SysML model than for the communication purposes of a diagram per se — after a certain point, the semantics around the diagram is for the machine, not for people, especially at the scale here, even if the mathematical relations used for the link analysis are relatively straightforward. Note that one quirk of the environment used requires that certain

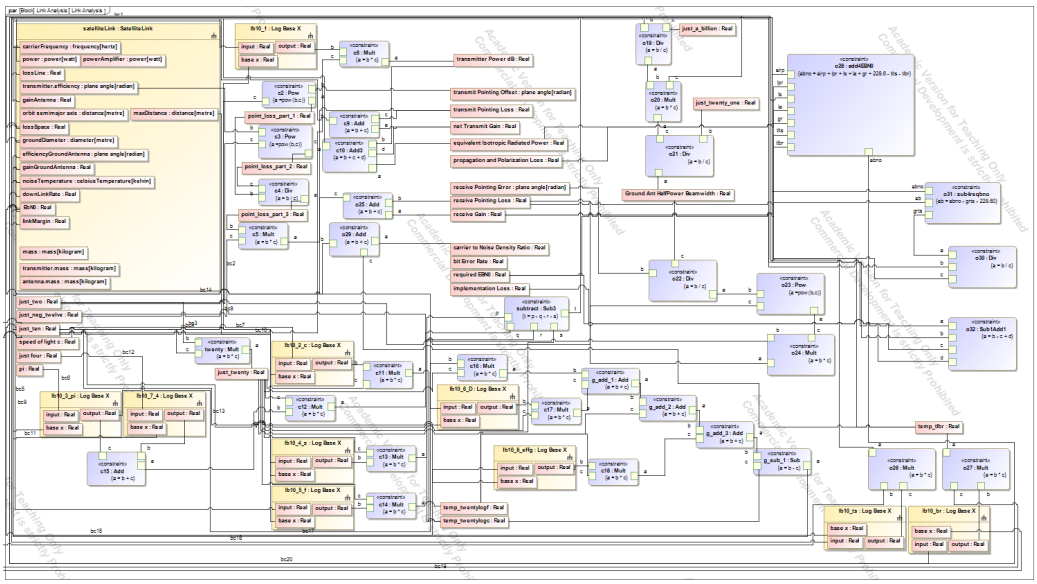


Figure 11.4: Parametric Diagram Represented Constraint Graph

relationships are broken up into smaller pieces, in the attempt to enhance re-usability but also for the algebraic solver. This means that to recover the logarithmic sums and differences found in Wertz, one must follow the edges of the binding connectors across Figure 11.4, as the relations have all been exploded out into many reusable units of engineering mathematics. However, generally this price is worthwhile, as the result is a bidirectional analytical model for the system of interest according to the causalities which may be declared across the parameters. Visualization of the network can be improved by various utilities[131, 132], and thus result in Figure 11.5.

At this point, many of the central issues might have been resolved, and the criteria should be revisited to see if this model might satisfy the needs for enabling robust design.

11.2.5 Evaluation of Criteria

Above, the baseline representation for the analysis in the system model has been established. This representation follows a constraint-graph paradigm of modeling and provides the ability to explore the design space bidirectionally by varying parameters.

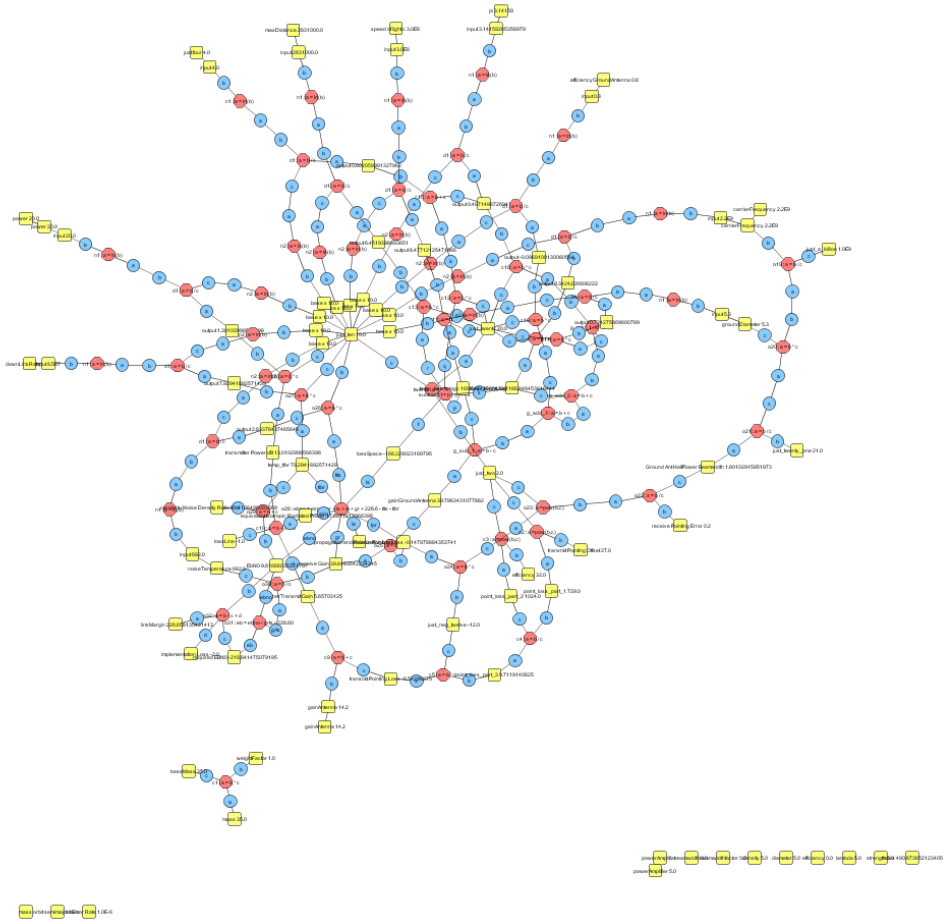


Figure 11.5: Buzztoys Panorama representation of the Instance Model for the CBAM using the Parabolic antenna for the antenna value

Now, the criteria established above will be assessed against the representation and its software ecosystem. To recap, the criteria where:

1. Ability to specify distributions for input or given quantities
2. Ability to automate simulation of many cases
3. Ability to analyze response data visually or otherwise
4. Ability to store data if necessary for future study
5. Ability to scale the approach to analyses of higher fidelity

According to Bajaj et al[130], this environment is quite capable of running the simulation over many cases. However, the trade study or Design of Experiments (DoE) is described in a spreadsheet software, row by row, and not in general about the DoE structure in a parsimonious manner. Additionally, while some of the tools in the ecosystem described by Bajaj may be capable of monte carlo simulation, less clear is the extent to whether probability distributions are handled in the description of the parameters, and monte carlo simulation is not necessarily a primary feature of the environment used above. Furthermore, any usage of probability might need to be constructed by external, custom functions linked to the tool as shown in some cases by Bajaj. Bajaj's ecosystem may provide visualization capability; however this capability will be outside the system model. Data however can be stored in the system model in instance specifications, yet caution should be taken if the trade study involves megabytes of numerical data. Indeed, the approach scales to higher fidelity, again as illustrated by several of the related works [130, 129]. Thus, the overall ecosystem is very capable — recalling before the dependency of these capabilities on software features. However, the particular representation in the end does not do much of these things. It has no direct representation of probability, any usage of a trade study is not necessarily parsimonious and does not bear much definition with the model, and in any case much of the data is stored outside the system model in various documents and files necessarily. As discussed for Gross but clearly demonstrated here, there is a gap to improve by providing direct representation and generation capability regarding probabilistic methods and DoE, thereby providing greater extensibility for the analytical representations in a system model.

11.3 RDS Implementation

As discussed in Experiment 4, RDS consists of several steps which have some aspects in common. That is, three steps involve the creation of DoEs for various purposes

(screening, combined array, and cases for monte carlo) with repeated creation of surrogate models (response surfaces from the combined array data, response surfaces of the distribution parameters from monte carlo[71]). The approach here will not be to capture every possible step of the process. Instead, the key features of the process will be the focus, i.e. specific enablers, which help address the criteria listed above. That is, features regarding DoE and monte carlo via intermediate representations. First of all, an analytical environment is required which can generate a DoE, calculate a disciplinary analysis such as the link budget analysis, and which can do monte carlo simulation. Due to interface definition in preparation for the intermediate representation these environments might be runnable independent of the system model. The key focus will be: creating more than one kind of DoE, and applying monte carlo analysis simultaneously to a DoE. As the link budget analysis is simple, it is very rapid and metamodeling or surrogate modeling can be ignored; however, the format used can be wrapped in a style befitting increased fidelity — noting of course that the fidelity to be used here is the same as the constraint graph representation above. The three capabilities — DoE, Monte Carlo + DoE, and running the cases through the analysis — will be interfaced to a system model representation and realize Hypothesis 4.

11.3.1 Design of Experiments Generator

The first aspect of implementing RDS is the need to handle multiple kinds of DoE, whereas previously only a single DoE was used in Experiment 1. This is due to RDS including different kinds of DoE, such as for screening or for running cases for Response Surface Methodology (RSM). The result is a generalization of the Experiment 1 implementation using PyDOE2 and and transforming JSON formatted inputs to a comma-separated textual output. This implementation, as it goes from textual arguments to textual result, can be run in the command line directly standalone or as

part of a pipeline. This is an important feature for reuse of the DoE generation capability, so that it can be run in different scenarios without modification to the source. The kinds of DoE available include generalized full factorial, 2 level full factorial, fractional factorial, plackett-burman, box-behnken, and central composite designs.

Screening

Screening involves running a DoE focused on detecting main effects. This could be a Taguchi array, a 2-level fractional factorial design, or a similar DoE which attempts to detect the main contributors to the analytical model's response values. The amount of contribution to the response values can be deduced through a variety of techniques. Firstly, the difference between levels can be plotted one factor at a time. Secondly, an Analysis of Variance (ANOVA) can be conducted to illustrate the percent contribution and statistical error, as well as considerations for pooling factors and interaction studies if interactions are considered in the DoE.

Combined Array

The combined array provides one DoE for the control factors and a second DoE for the noise factors. The analysis is performed by running each row of the noise array per row of control array, enabling averaging of the various responses per control array row. This technique enables varying noise, environmental, or other less-controllable factors to detect the variability of the model. For example, in the case of FireSat communication system, the amount of loss due to atmospheric water might be modeled as a noise factor with levels bounding the expected losses. Conducting this sort of combined array would target a design of the communications system with more robust performance (more consistent performance) with respect to water vapor losses in signal. However, there is an important pattern at work here. First of all, the combined array involves running one case from one DoE for each case in another

DoE[71]. However, while one of the two DoEs is for the noise variables, no probability is yet introduced — probability being a top objective of this present experiment. Furthermore, a simple concept of sampling the probability distributions would be to do so for the number of times required by the monte carlo analysis (e.g. 10, 100, 1000, 10000) for each case in the DoE of the control factors. Thus the pattern of the program is the same, involving some sort of double for-loop at least in the simplistic approach. Therefore, the priority in implementation as discussed above is given to the DoE with Monte Carlo, and this specific capability is skipped for the purposes of the overarching objectives related to Hypothesis 4. However, note that it is possible to revisit in a similar manner to the monte carlo analysis.

DoE with Monte Carlo — Random Variables

The Monte Carlo analysis is the premier capability, and in particular the representation of inputs for the monte carlo analysis. For RDS, Monte Carlo simulation is performed for each entry in a DoE. The data for the entry in the DoE is abstracted by fitting a gamma distribution (or perhaps similar, but for simplicity, taking the literal approach from [71]). A slight difference in detail between [71] and the libraries used for this capability (scipy.stat, as used for exponential distribution fits in previous experiments), is that the gamma distribution is represented by a three-tuple (shape, location, scale) rather than a two-tuple of (shape, scale) for each distribution. This may not be a material difference for example if somehow the original study were centered around zero or otherwise such that the location parameter were irrelevant. That said, success here is the ability to generate these parameters for each case of the DoE such that the RDS process might continue with a manual effort to fit surrogate models to the distribution parameters and conduct the optimization and constraint studies. One note however on implementation, is that the monte carlo simulation and random variable representation will be handled in the subsequent

alongside the analysis representation, rather than as part of the DoE. Random variables are currently permitted to have uniform, triangular, or normal distributions. More distributions are easily added to the environment code.

Summary of DoE Generation

The DoE generation proceeds according to command line arguments, specifically JSON information describing the prescribed DoE kind of the DoE kinds implemented as well as the factors of interest and their levels (minimum and maximum). The program itself was implemented twice, once for complete standalone operation and once for use with the system model. The standalone operation variant returned the text of the DoE directly to the "stdout" of the command line, so that the DoE generator might be stitched together over Unix shell with the analytical environment, and was tested specifically on Antenna design. The antenna design example demonstrates that the DoE Generation was successful and Analysis of Variance was possible according to the outputs, and did indeed show some predictable results, such as the importance of the diameter of the reflector. The second variant of the DoE generator provided a file path to a CSV file which stores the contents of the DoE settings generated in a location provided as command line input, alongside a name for the study as an additional input variable. Figure 11.6 illustrates the results obtained, showing success of the screening DoE — these results should be taken with caution due to confounded factors and are not representative of a baseline from e.g. SMAD 2003[133]. Of course, the primary focus of this experiment is not to build the best analysis possible of parabolic reflectors. Instead, it is regarding the capability related to Hypothesis 4.

Regardless, the DoE capability is effective, and the antenna gain will be posed as an input to the link analysis as an exercise in setting domain requirements. Furthermore, the mappings from the system model are still required, beyond shell scripts

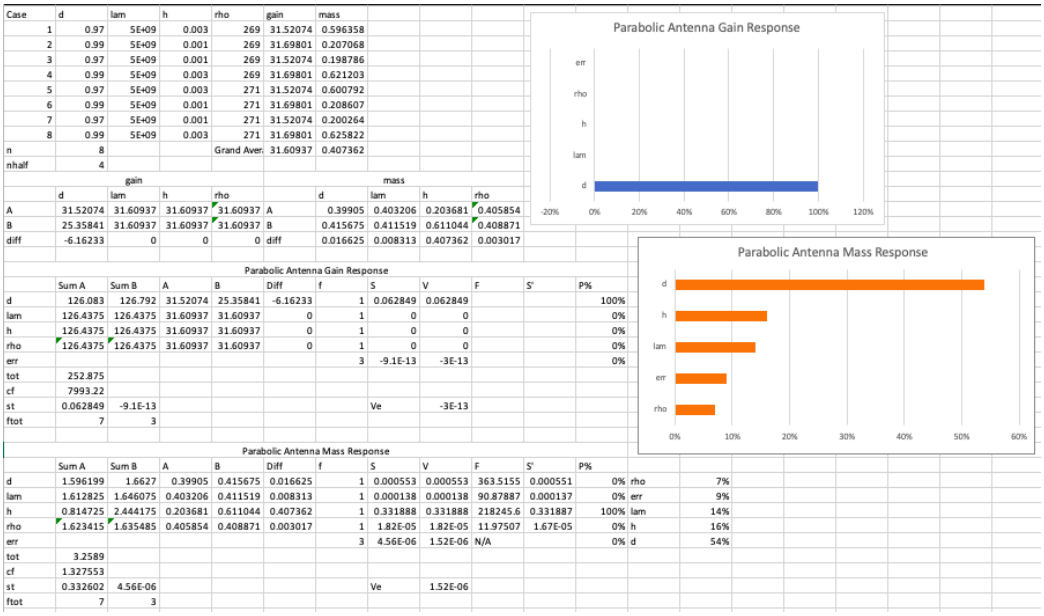


Figure 11.6: Parabolic Antenna ANOVA illustrating success of the Screening DoE capability from the DoE generator; results show potential issues in confounded factors.

and basic computer automation.

11.3.2 Analysis Capability

The analysis capability includes the ability to generate response data for various satellite design routines. This includes antenna gain for two kinds of antenna as well as the satellite link budget, as a missing piece in the FireSat analytical description in Friedenthal and Oster[98] while prominently featured by Gross and Rudolph[128]. The antenna designs included are parabolic and horn. The outputs of antenna design in this case are the natural log of the gain as well as the estimated mass of the antenna. Antenna gain is an input then to the link budget analysis, rather than as a coupling variable between relations on beamwidth and net gain. One limitation not yet mentioned of the constraint graph approach is that if creating a CBAM for the antenna gain analysis for the Parabolic antenna, the causality labelings mean that the instance specifications utilized are not directly reusable for the link analysis CBAM instance model and the causalities must be reset/redetermined. This means

that to do multiple CBAMs without refactoring the model for a specific formulation of composition (i.e. a third CBAM), that the instance specifications must be copied and pasted after doing the analytical activity so that, should the previous analysis need to be revisited, work is not lost to resetting of causalities and variable quantities. This issue is another motivation to decouple the analysis and take antenna gain as an input to the link margin, rather than connecting them together, as they could not be connected truly in the constraint graph approach but need be done sequentially — designing first the antenna and then the link if using two CBAMs. Here, the approach might be seen as determining feasible bounds or side constraints for the antenna gain based on the link analysis or even perhaps eventually a sort of inverse design activity may be possible by leveraging the products of RDS[134]. Importantly, the link analysis should be as correct as feasible, and was shown to be reasonable with respect to the baseline figures in Wertz for a link margin greater than 3 dB. However, to do this, the rest of the mappings will be established. Source code for the system analysis and DoE capability can be found in Appendix A, Section A.4.

Establishing The Mappings

Peak et al 2010 describe native interfaces, standards, and other capabilities as providing some interface or integration between software tools in an environment[132]. The claim in this document is not that such a statement is incorrect, but rather about a strategy on how to go about creating customized interfaces of this kind for analysis integration via intermediate representations[64]. Specifically, this experiment proposed two mappings L and M such that:

$$\Gamma_{S_v} \underset{L}{\mapsto} \Gamma_{I_v}$$

$$\Gamma_{I_v} \underset{M}{\longmapsto} \Gamma_{E_v}$$

In fact a version of these mappings has already been used in Experiment 3. However, for the purposes of establishing as concretely as possible without a deep dive into source code, the following discussion will illustrate the mapping relations in terms of where data originates and where it goes to for mappings L and M. The focus will apply to the noise variables in particular for the probability discussion of Hypothesis 4, yet similar mappings will be constructed for DoE factors, DoE kind, constants, and potentially other aspects of the analytical environment.

Firstly, the standard that will be used as an intermediate representation between the system model and the analytical environment is JSON. Additionally, components of the analytical environment may require additional variables provided as command line arguments one at a time, as text or as integers as appropriate. Furthermore, the analytical components may also use or provide inputs which are in comma-separated format, being slightly more efficient in memory than JSON for large datasets. However, as the objective is a parsimonious representation of probabilistic methods as a labeling on the SysML model, such large datasets will be left outside the system model. Figure 11.7 illustrates the first mapping, L, from SysML to JSON for the noise variables. Note that the full implementation of the mapping L is given in Appendix C, Section C.3 and is slightly modified from the Walworth mapping in Section C.1. The modifications enable this variant to be utilized four times in the simulation process, shown later in Figure 11.13.

Figure 11.7 shows an example of the block definition for a particular uniformly distributed variable. This variable is realized by an instance specification on the right. Various aspects of the data in the SysML model are transformed. The part property name is a key for the JSON object describing the variable. The data from the instance specification is used to populate fields in the JSON similar to what is seen in the SysML diagram. Finally, an additional field specified by a value property through subset relations is seen for the distribution parameters, with ordering that

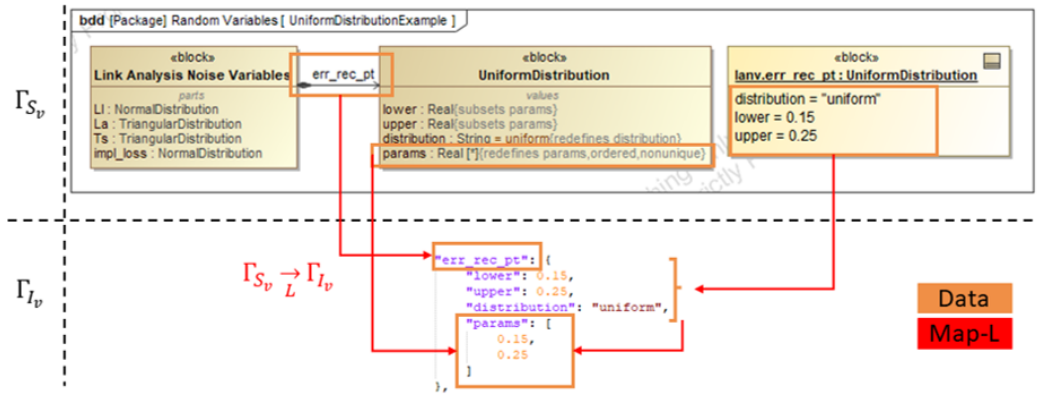


Figure 11.7: Mapping L from SysML to JSON

ensures the values are available in automated fashion for setting up distributions. In this case, the SysML "labeling" is a kind of class hierarchy specifying the available distributions which might be used to determine the noise variable inputs in a sort of variant of the CBAM concept as described by Contribution [62, Paper B]. The hierarchy is illustrated in Figure 11.8. This hierarchy is directly used in creating the noise variables.

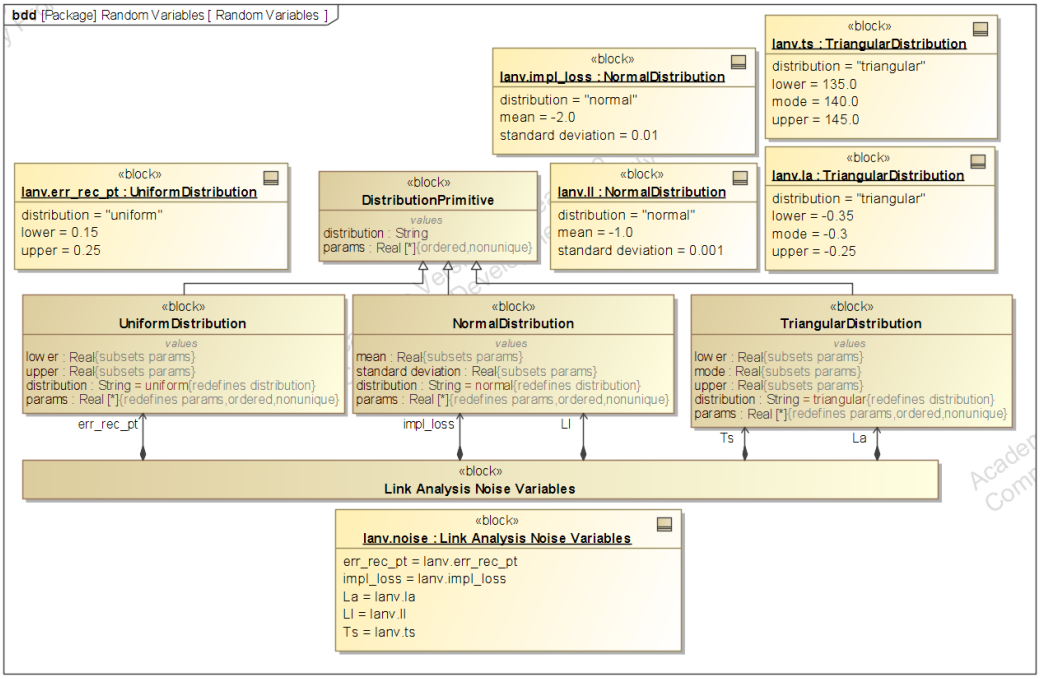


Figure 11.8: Noise variable definitions using distributions

This way, for a specific set of analysis inputs, the distributions can be applied to the particular noise variables by part property. As seen in Figure 11.8, it is possible to reuse these distributions in specifying noise variables, and this holds not just for one set of noise variables but in practice for any set within the model. Note that such a representation is probably do-able with the constraint graph approach given sufficient software development support as discussed earlier, however in this case the effort has been invested to directly obtain the distributions by a subsequent step. That is, the map M as shown by Figure 11.9 which specifies how the data maps from JSON to Python.

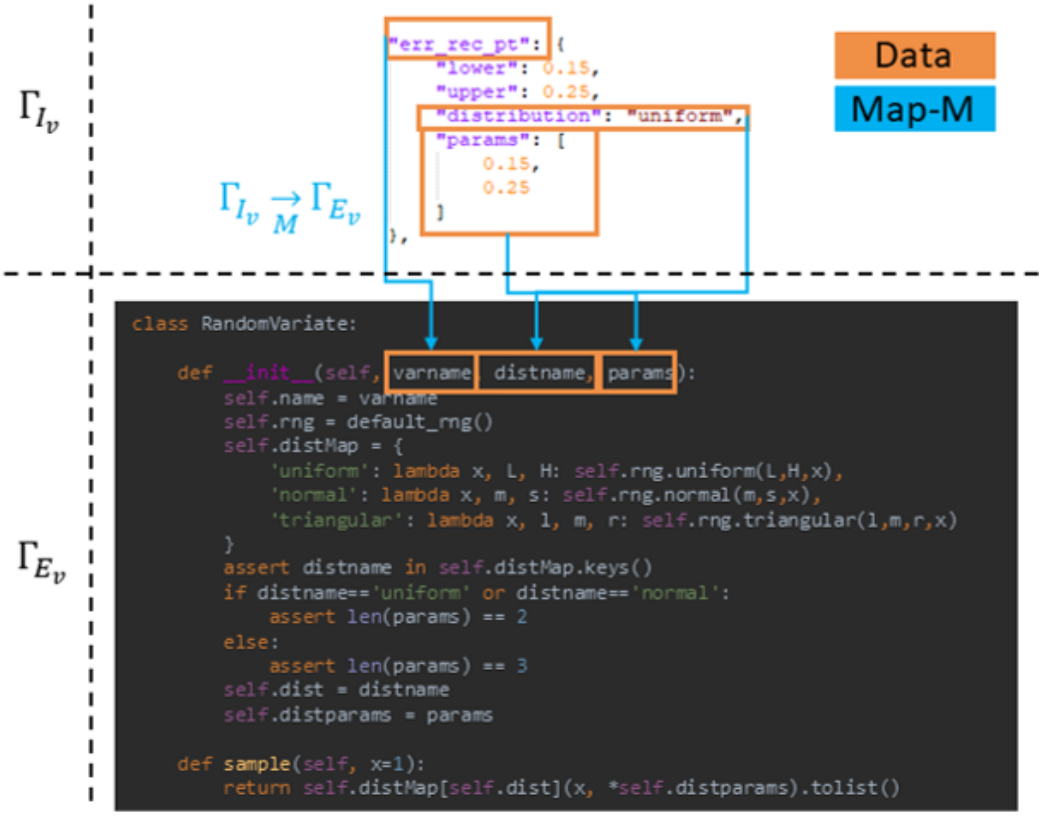


Figure 11.9: Mapping M from JSON to Python

According to map M, the data is fed into the Python class definition for a `RandomVariate` which specifies the distribution that can be sampled x times for monte carlo simulation. Note that in the Python implementation, the available distributions are directly

listed and if additional distributions are added, they can also be represented in SysML and thus available for use in noise variable specification. However, as established now, the currently available distributions are ready for use. One important note is that these maps are not defined purely on vector spaces — recalling the discussion of architecture exploration in Experiment 4 vis-à-vis Sharma[122]. The mappings from the architecture description (i.e. SysML) to the analytical environment are done across a series of steps to access and transform the data via the intermediate form. Any real mapping from a language like SysML to an analytical environment is unlikely to be such a linear map in vector spaces outside of specific circumstances, e.g. what might be encountered in certain constraint parameters as seen in Bajaj et al[130] where multiplicity results in vectors by specific interpretation of complex aggregates by the tool. Much more common are these complicated mappings whereby data from the SysML model is systematically modified for use by the external environment. By using these intermediate mappings, custom approaches for particular domains are facilitated by simplifying the overall process and leveraging some commercial tools and the basic features of the SysML environment. Furthermore, the process underlying L attempts generally to preserve structure from the SysML model. The process may be tuned to accept or reject items available in the SysML model, such as constraint properties, constraint parameters, enumerations, reference properties, etc. One difficulty which may arise are recursive relations due to non-directed associations, and these would require additional constraint which may limit the extent of structural preservation. However, for the case shown here, the structure of the SysML model is preserved in transformation to JSON but much of the information is filtered such that just the specific content of interest is transmitted externally. A more mathematically-oriented thesis might then formalize this map in terms of functors between categories of SysML, JSON, and Python. For the purposes here, no such formalization is claimed, given the practical considerations which often arise from one implementation to the next.

For example, the mapping L handles listed items differently here in Experiment 5 than the similar mapping did for Experiment 3, which was appropriate as far as the testing and exercise of those mappings for their purpose. Such specific implementation concerns likely impede a mathematical formulation but are important to the practical workings of affecting the result desired via Hypothesis 4.

The Greater Analysis Representation

The analysis representation otherwise bears many similarities to the approach taken with Walworth et al. The primary difference is a grouping of the various portions of the input space by DoE factors, noise variables, and constants. The specification is shown at a high level in Figure 11.10. Zooming into the inputs for a moment,

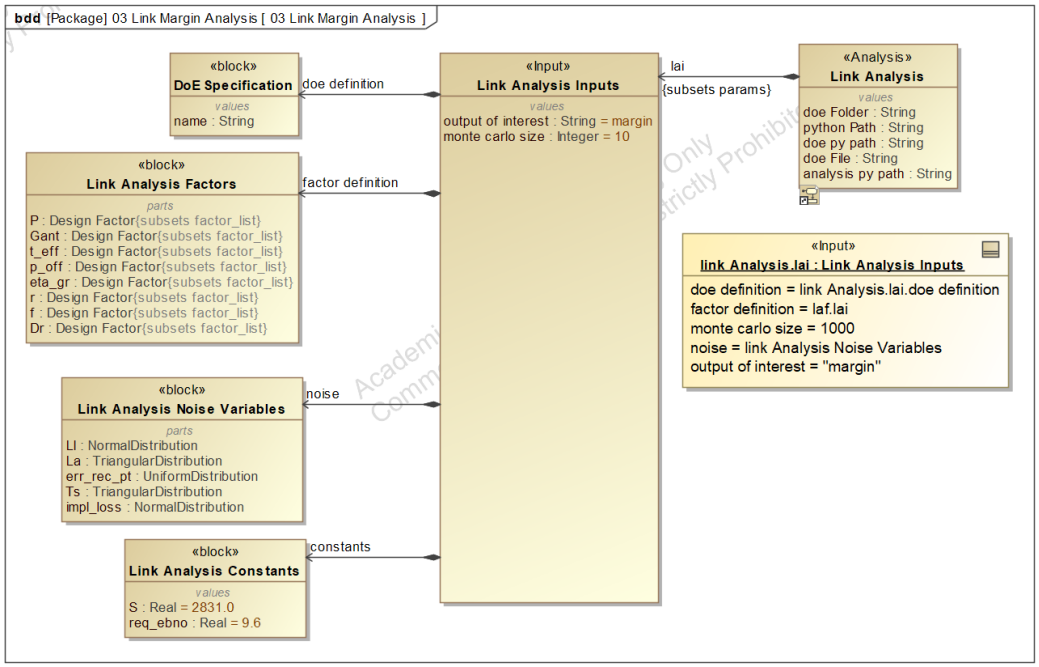


Figure 11.10: Link analysis definition

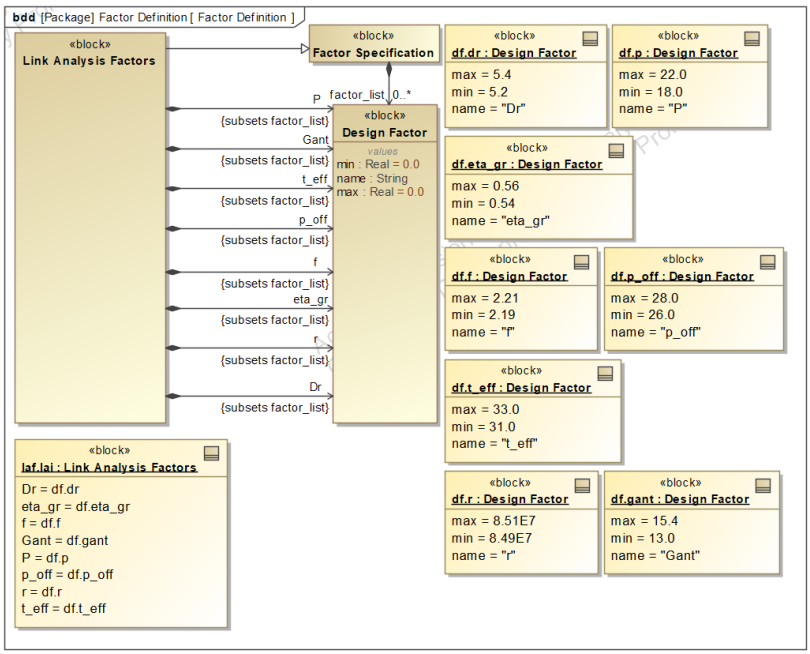
having already covered the noise variables above, there are several other items of interest. For one, the monte carlo analysis is to be performed on a specific output of interest, listed in the inputs for easy manipulation. Secondly, the number of runs for the monte carlo simulation is available as a simple integer to toggle for the analysis

case. Constants are defined in a block and subject to a similar mapping as the Figure 11.7 but focused primarily on the simple value definitions within the constants block. The specification of the DoE factors is complex similar to the noise variables. Figure 11.11a illustrates the structure available for specifying DoE factors and how this is done for the particular choice of DoE factors in this example; if a part property is removed from link analysis factors and a value property with the same name is added to constants with appropriate default value, then the analyst will have changed the definition of the DoE for the analysis case.

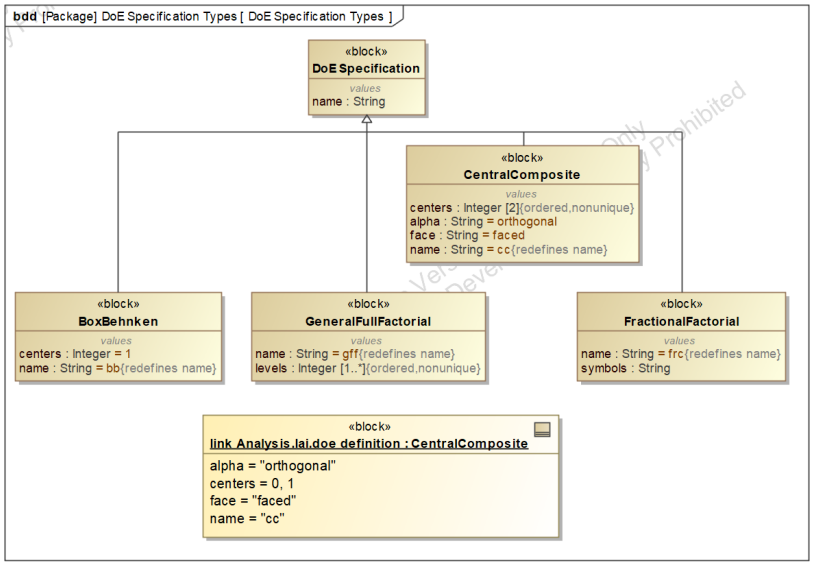
Finally, some addition information is required to describe how to construct the DoE based on the selected factors and levels. For this purpose a set of DoE definitions are created, illustrated by Figure 11.11b. For the DoE generator, some simpler DoEs can be selected by using a DoE Specification instance with the appropriate name. This covers, for example, plackett-burman. For the DoE specifications which require a bit more information, they are defined as shown, alongside a specific example for the analytical case of a face-centered central composite DoE — two of which were used in Mavris et al[71] including as part of the monte carlo analysis phase for RDS.

In order to drive the analytical process, a series of SysML activities are used to generate the DoE and run the monte carlo simulation. In this set of activities, the map L is run a total of four times, once each for constants, DoE Factors, DoE Defintiion, and noise variables. Figure 11.12 illustrates the overall two-step process. Figure 11.13a is the process by which the DoE is generated from the input information, and the result is a file path to the CSV file with the DoE factor levels (non-normalized for use in the analysis). Then, Figure 11.13b illustrates the subsequent process to run the DoE for Monte Carlo simulation as well as providing the additional input detail. Output data is stored in two files, for purposes that will be discussed shortly.

When configuring the behaviors for simulation, it is useful as in the constraint graph approach to rely on instance specifications for saving the data related to indi-



(a) DoE Factor Specifications



(b) DoE Definition

Figure 11.11: Additional Definitions

vidual cases. For example, if in one usage of the analysis 1000 monte carlo runs are desired, while in another 10000 are desired. Additionally, there are the distribution parameters and DoE factor settings for the input variables. Figure 11.14a illustrates the DoE Factors used in the current study, for the purpose of illustrating that the

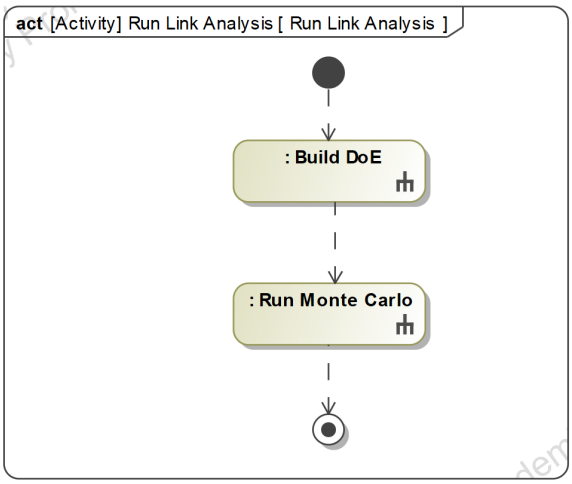
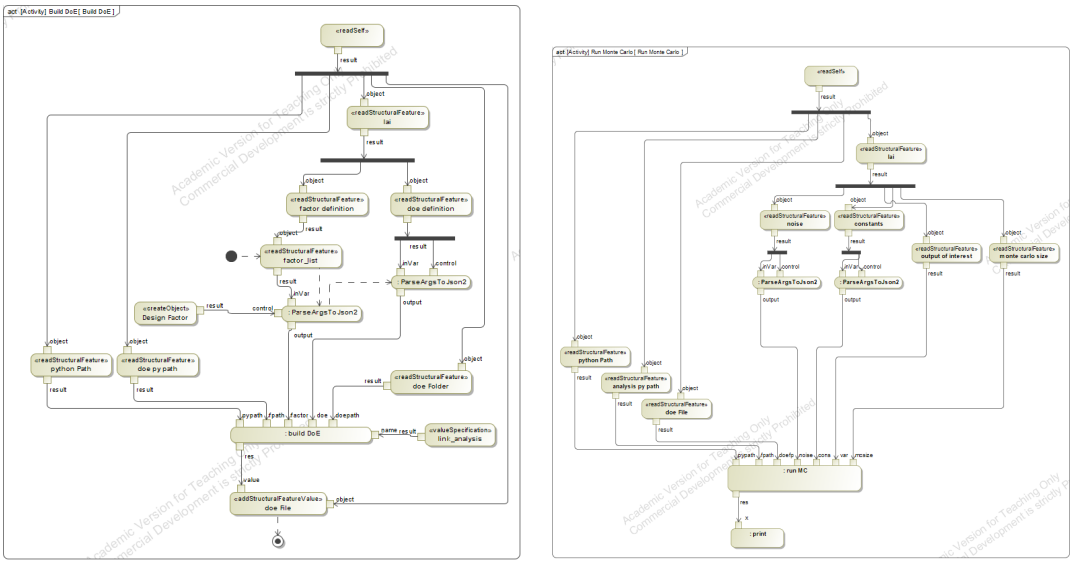


Figure 11.12: Overall process to run the analysis



(a) DoE Generation Process (b) Monte Carlo simulation process

Figure 11.13: SysML Activities Describing Analytical Process

analysis bounds the design presented by Wertz. Additionally, the noise variable distribution parameters for the particular study are illustrated in Figure 11.14b.

11.3.3 Results and Study of the Data

The face-centered central composite design for eight factors results in 273 cases for the DoE for the link analysis. Factor levels are chosen to bound the FireSat payload

#	Name	name : String	min : Real	max : Real
1	df.dr	Dr	5.2	5.4
2	df.f	f	2.19	2.21
3	link Analysis.lai.factor definition.eta_gr	eta_gr	0.54	0.56
4	link Analysis.lai.factor definition.gant	Gant	13	15.4
5	link Analysis.lai.factor definition.p	P	18	22
6	link Analysis.lai.factor definition.p_off	p_off	26	28
7	link Analysis.lai.factor definition.r	r	8.49E7	8.51E7
8	link Analysis.lai.factor definition.t_eff	t_eff	31	33

(a) DoE Factor Specification Instance Table

#	Name	distribution : String	lower : Real	mode : Real	upper : Real	mean : Real	standard deviation : Real	lower : Real	upper : Real
1	link Analysis Noise Variables.err_rec_pt	uniform						0.15	0.25
2	link Analysis Noise Variables.impl_loss	normal				-2	0.01		
3	link Analysis Noise Variables.la	triangular	-0.35	-0.3	-0.25				
4	link Analysis Noise Variables.ll	normal				-1	0.001		
5	link Analysis Noise Variables.ts	triangular	135	140	145				

(b) Noise Variable Distribution Parameters

Figure 11.14: Instance Tables for Analysis Configuration

data performance described by Wertz, and distributions were set for noise variables to keep their values similar to the baseline as well to ensure that output quantities were reasonable. For the monte carlo simulation, 1000 runs were applied. As described earlier, the process for running these cases is to do each of the monte carlo simulations on each case in the DoE. So for the current DoE row, with the constants, sample each of the noise variables and run the analysis for the specified number of times. All input settings (including noise variables post-sampling) and output results are saved, resulting in 273000 cases. Note that for the simplistic analysis representative of the link budget or link analysis, the model runs 273000 cases in approximately a minute or two while single threaded on a R5 3600 CPU with 16 MB of RAM. For this large set of cases, the DoE case number as well as the monte carlo iteration are also recorded. Each DoE row's 1000 samples of the margin variable are fitted to a gamma distribution as the output variable of interest, and the distribution parameters are also saved, resulting in 273 results each of the shape, location, and scale parameters described earlier. Results are saved into two CSV files at the end of the simulation process. One CSV contains the raw output data, all 273000 rows as specified here,

and is approximately 100 MB in size. The other file, containing the distribution parameters for the 273 DoE cases on margin, is more reasonably sized. If following RDS precisely, these distribution parameters would be used next to create surrogate models of the distributions to do optimization and/or constraint plotting against the probabilities. However, because the raw data is also exported, this data can be viewed and manipulated. For a link budget, the rule of thumb for link margin is often that it be greater than 3 dB.

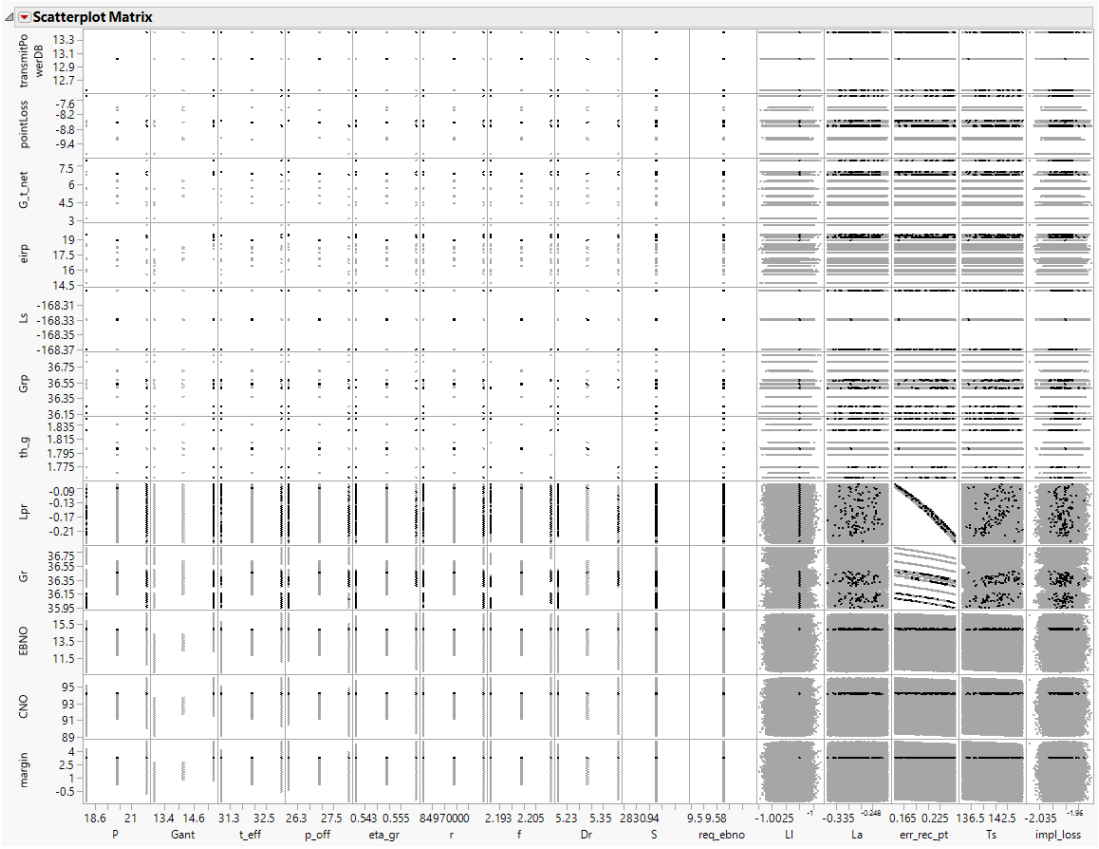


Figure 11.15: Scatterplot matrix in JMP illustrating all data from all samples

Figure 11.15 is a scatterplot matrix that enables the user to explore the designs represented in the results. It is possible to select specific points and further explore their parameters to see the input values which resulted in particular outputs. In particular, from the standpoint of analytical validation discussed in the literature review, this view shows the span of data based on running the DoE around the values

defined by Wertz. As a result, some cases might be infeasible, and some cases are much more performant, but many cases are approximately in the same domain of performance as the Wertz FireSat link budget. Thus, the analysis is at least well-calibrated towards the baseline and additional studies on a wider space might be performed. In Figure 11.15 a swath of points are selected, representing the band of designs which are around the link margin of 3 dB, notably less performant than Wertz but perhaps an acceptable boundary for the feasible space which in the last row, for margin, lies above this band of designs. Further study could widen the design space. Additionally, on the subject of requirements validation, the antenna gain input now provides bounds on what should be expected from the antenna design. Regardless of technology, the necessary antenna gain space can now be specified or bounded accordingly. Clearly, the antenna analysis illustrated earlier is faulty, and this technique has assisted in validating the link analysis while invalidating the quick and simple antenna analysis used before.

Additional insights are possible with this data. With these tools, a highly interactive stage of the RDS tasks might begin. For example, fitting surrogate models, creating constraint plots, prediction profilers to view sensitivities, etc, and of course focusing on those gamma distributions. One outcome might be to explore changes in the data rates, other input variables, or apply constraints on other outputs besides the link margin. These tasks are not new, and work like them has been presented many times since Mavris et al[71]. However, the key advancement in this chapter is to provide enablers towards DoE, probability, and running analyses via the system model to assist in the steps of RDS and to improve the understanding of requirements during the requirements validation phase. These capabilities are clearly enabled by such an environment for design space exploration, and provide strong support for Hypothesis 4.

11.3.4 Evaluation of Criteria

The criteria for evaluating the technique for exploring the design space were, again:

1. Ability to specify distributions for input or given quantities
2. Ability to automate simulation of many cases
3. Ability to analyze response data visually or otherwise
4. Ability to store data if necessary for future study
5. Ability to scale the approach to analyses of higher fidelity

In the development shown above for the link analysis, distributions for selected inputs or noise variables are explicitly represented. The distributions are reusable and different analytical configurations might be established with different choices of noise variables and different selected distributions — and the potential for variability increases, considering the innumerable possibilities for the distribution parameters. Automation of cases is achieved by generating a DoE based on specifications in the system model, and running the cases on the DoE. Response data is visually analyzed in appropriate tools, as might be expected in the constraint graph technique. Analysis data is serialized to file types capable of handling large numerical datasets. However, as demonstrated with Walworth et al, it is not too difficult to bring at least some of this data back to the model if desired; however, the data should remain outside the system model if using a similar ecosystem and managing hundreds of megabytes of data. In this case, as shown here, the simulation is populating or using a DoE file path where all generated files can be found, including the DoE and the output data files. With a simulation configuration this path is easily stored to the slot represented the `doe Folder` value property, as a way to later look up the results for the case run via the instance specification. Finally, this approach can be scaled to higher fidelity,

as discussed or alluded to in Contribution [62, Paper B] and done by this author in other contexts which unfortunately cannot be published.

11.4 Conclusion

Overall, this implementation improves on various issues related to the constraint graph implementations discussed earlier and present in the literature as baselines from a variety of authors in the MBSE domain. The specific improvements target more the software features and intended design activities for use of the simulation model. When exploring requirements and investigating the validity of analyses, the application of statistics and probabilistic techniques assists in building confidence in the result. The probabilistic approach to design can further enable other advanced techniques in system design, and having provided this hook in the system model enables not only RDS as a requirements validation process, but various other related design methodologies. Of course, many of these activities require a person to be involved in the exercise of the technique, and not every facet of the RDS methodology is a candidate for automation. However, in this case, the objective was to close the gap between the dominant view of analysis in the SysML and MBSE literature with a technique that has wide acceptance in analytical communities. Further software developments in constraint graph parsing tools may enable them to perform the same tasks shown here, and it is important to note that the constraint graph technique is not rejected — merely, a different analytical technique is applied in order to develop the specific missing capabilities for Hypothesis 4 since these are not readily available via the baseline tools for constraint graph modeling in the ecosystem at hand, as previously described.

In conclusion, Hypothesis 4 is accepted. Intermediate representations enable a mapping from products of the system model to an analytical environment, and if the space of representations includes information about a DoE, probabilities, etc, then

probabilistic analysis that improves confidence in system design is enabled.

12

DISCUSSION OF RESULTING METHODOLOGY AND PLATFORM

This chapter will summarize and discuss the results of the experiments and how they impact the proposed methodology for P-SEMP.

12.1 Recap of Experimental Results

While not every experiment has gone to plan or as proposed, several interesting findings resulted.

12.1.1 Experiment 1 Conclusions

Experiment 1 resulted in probable rejection of the Hypothesis 1a. The main cause for rejecting Hypothesis 1a is the apparent mismatch between the capability and performance of the parameterized numerical models for requirements status, vs. actual practical descriptions of requirements status. Key issues arose in replicating Walworth et al due to the lack of an appropriately parameterized rework discovery "exposure rate"[108]. Despite these issues, a simple model can still be constructed by assuming some proportionality relation for the discovery rate. The simplified model lacks the jagged features of Walworth et al due to the non-existence of probabilistic effects. To this end, Experiment 2 may provide greater insight and promise towards the overall modeling effort: either by providing a rate calculation for rework discovery or by replacing the relation to work really done. Perhaps most important, the SD model(s) may remain critically important — not because of their parameterization or simulation capability, but because of the narrative that their visual portrayal can tell. This narrative capability and the motivating soft systems methodology may be the

primary benefit of Walworth et al, although Walworth et al do not go nearly as far as Lyneis 2007[111] in specifying extended causal loops for the learning system. While there are certain interesting questions — how does team experience and capability affect the time elapsed between requirement maturity elevation events, for example — there does not yet appear any means by which to answer these questions.

12.1.2 Experiment 2 Conclusions

Experiment 2 involved a particularly challenging software implementation. It is likely that an approach aiming at creating a compiler would be more flexible in simulation. The approach chosen is a compromise between the objective of a DES flexible in terms of SE methods, yet implementable in the time, skill, and scope of the thesis. As the simulation does not depend too much on the process description — other than in placing limits on mathematics such as uniformly distributed integers — many processes may be modeled and simulated by this tool. In the context here, only SE methodologies are of interest and the OOSEM-Lite process from [98] on Spacecraft Requirements Derivation is presented alongside SA-MD.

Overall, Hypothesis 2 is accepted. SE methods are clearly amenable to simulation by DES, as they are strongly driven by event-focused terminology. Further, as discussed in Experiment 1, SE methods and in particular requirements validation/requirements status is industrially dependent on event-specific markers. Unfortunately, these facts lead to the rejection of Hypothesis 3; the simulations for SRD and MD do not provide any quantification of error detection and therefore cannot be substituted within the SD model of Experiment 1. At best, these simulations may be replacements, as modeling a path from an initial state of work to be done to work completed, especially in the case of a specification like for SA-MD. In particular, the use of iterations for SA-MD and time (units unspecified) for OOSEM-Lite are interesting for task planning as they can be proxies for cost estimation in terms of person-dollars per

hour or person-dollars per iteration. Therefore, regarding Hypothesis 3, the bi-level simulation is what is rejected; utility is still expected for project planning, especially upon integration with a system modeling environment.

12.1.3 Experiment 3 Conclusions

Despite the shortcomings of Experiment 3a, Experiment 3 shows much success via Experiment 3b and the specification of method models in the system model for analyzing the SE method proposal. While the Walworth example is typical of more routine analyses, the DSL constructed for the DES representation is a potential contribution of this work and provides another means for simulation. Beyond the specific target applications of SE planning, other DES using the actions posed by Experiment 2 may be possible via the system model, and most importantly other than the system modeling tool itself, the core simulation environment is constructed using open-source software. Thus, while many similar techniques are significantly locked down, this technique may be extensible to other system modeling alternative environments. Insofar as the DSL implementation illustrates the promise of system model customization for domain analysis, Experiment 3 illustrates the practical capability of these languages beyond representing semantic information underlying SE documentation. While Hypothesis 3 is for the moment rejected due to the lack of bi-level simulation available in Experiment 3a, environments such as that constructed here in Experiment 3b answer Research Question 3a by enabling the exploration of parameter values in close coordination with the system model environment, thus enabling better exploration of the process, schedule, or other design space as documented and managed within the MBSE ecosystem.

12.1.4 Experiment 4 Conclusions

Experiment 4 aimed to prove Hypothesis 1 as to whether the over all platform for SE modeling and planning could enable decision-making on the SE method proposals for SE processes according to measures, such as leading indicators, which may be correlated with lagging indicators such as cost and schedule. In this case, While the targeted leading indicator model can be exercised, it does not produce much insight. However, the method models generate substantial information, characterize method performance, provide probabilistic assessment, enable setting targets and process requirements, and provide the capacity to study variability in method specification both in terms of the limits applied to variability as well as the task network itself. Furthermore, unlike traditional forms of analysis, as discussed in Experiment 2, these task networks can include cycles. After a discussion on architecture exploration, a mathematical model was used to demonstrate that the underlying theory of Experiment 2 and 3 performed this kind of exploration, and to show that by changing the method specification content, vastly different performance would result. These changes in performance were shown to be realized both by tightening performance on the existing network, as well as changing the task network itself. Furthermore, additional criteria of subjective or qualitative nature were acknowledged and established to provide a broader picture of the method proposal consideration. Then, to illustrate constructively how the DSL of Experiment 3 enables method proposal exploration, a third method RDS was represented and simulated from the system model. After a brief discussion on how the method proposal simulations relate to cost per Experiment 2, the overall picture of the methods and criteria or variables was given by Table 10.4. Systems Engineers and project planners can look at this table, apply their preferences, and perhaps make some decisions or even revisit the simulation in the system model to compute new performance expectations, all while remaining within the system model and incorporating SE planning in MBSE. Altogether, Hypothesis

1 is accepted as the platform established across Experiments 1, 2, 3, and 4 enables the exploration of SE methods via SE task performance prediction, and brings all the information together in the system model for modeling and planning purposes.

12.1.5 Experiment 5 Conclusions

Experiment 5's implementation for noise variables, DoE, and running the link analysis accordingly per key steps in RDS improves on various issues related to the constraint graph implementations discussed earlier and present in the literature as baselines from a variety of authors in the MBSE domain. The specific improvements target more the software features and intended design activities for use of the simulation model. When exploring requirements and investigating the validity of analyses, the application of statistics and probabilistic techniques assists in building confidence in the result. The probabilistic approach to design can further enable other advanced techniques in system design, and having provided this hook in the system model enables not only RDS as a requirements validation process, but various other related design methodologies. Of course, many of these activities require a person to be involved in the exercise of the technique, and not every facet of the RDS methodology is a candidate for automation. However, in this case, the objective was to close the gap between the dominant view of analysis in the SysML and MBSE literature with a technique that has wide acceptance in analytical communities. Further software developments in constraint graph parsing tools may enable them to perform the same tasks shown here, and it is important to note that the constraint graph technique is not rejected — merely, a different analytical technique is applied in order to develop the specific missing capabilities for Hypothesis 4 since these are not readily available via the baseline tools for constraint graph modeling in the ecosystem at hand, as previously described.

In conclusion, Hypothesis 4 is accepted. Intermediate representations enable a

mapping from products of the system model to an analytical environment, and if the space of representations includes information about a DoE, probabilities, etc, then probabilistic analysis that improves confidence in system design is enabled.

12.2 Summary of Key Developments

Several key developments have resulted from the P-SEMP thesis.

12.2.1 Affirmative Findings of the P-SEMP Thesis

Affirmative Findings are the positive findings of the P-SEMP thesis which indicate what should be done, or confirm various aspects of the proposal, proposed Platform version 0.1 in the conceptual architecture, and associated P-SEMP methodology.

The experiments in the P-SEMP thesis affirmed the application of DES for analysis of proposed tasks in SE efforts which constitute the proposed methods for SE processes in SE methodology. This affirmation included the development of a flexible simulation environment in Python using simpy, which ingests simple specification files describing the proposed methods. These methods may have tasks which proceed linearly, branch conditionally, or which experience non-deterministic iteration. The simulator tool has the ability to work in concert with a system model — a new customization of SysML provided a domain specific language which enables planners to record their description of the proposed method and tasks whilst the language itself uses that description to set up the data necessary for the input specification to the simulator. A simple analysis context can then feed the pre-configured data to the simulator from the system model, enabling exploration of task performance based on the record in the system model. Example simulation specification input models are listed in the Appendix as A.9 and A.10, while the customizations which enable the system model to build the data for these kinds of specifications is illustrated in Figure 9.12.

Additionally, supporting the concepts of this simulation environment, a new cri-

terion was established specifically targeting whether the non-deterministic iterative task/method proposals will halt. The formulation provides an alternative analysis technique for rapid studies, which in the case of the iterative task specifications is perhaps especially important to save compute time later (should one attempt to simulate a never-ending process). This criterion added additional parameters — though, as graph or architecture parameters, they are discrete or disjoint across alternatives — which could be recorded and compared across alternative method proposals and this was done in the final Table 10.4.

Finally, there are the contributions towards representing probability and designs of experiments, which uses the analysis representation style shown earlier in the thesis alongside external tooling built to support the set of mappings by which the system model specifies the distributions and cases for combined DoE and Monte Carlo in RDS, tackling the key pain points of that technique as far as tools for use in SE methodology. While other smaller contributions may be notable (simple error detection, requirements revision data, etc), this example of how to implement such capability in a system model demonstrates the utility of such an approach for system design.

12.2.2 Negative Findings of the P-SEMP Thesis

Negative Findings are aspects of the proposed Platform version 0.1 which were shown to be, if not incorrect, at least dubious. These findings have value in informing engineers what they might avoid if their interests align with the Platform version 1.0 phenomena for analysis; specifically, proposed SE methods and planning impacts which fall out as a consequence. The strength of these findings has been more variable than in the Affirmative Findings.

Two negative findings resulted from the experiments of the P-SEMP thesis. The first was regarding the use of the Walworth et al model as a baseline for a require-

ments status or requirements volatility leading indicator model. For this particular model there were two issues. First, the relation for the implemented Jelinksi-Moranda exposure rate was unrecoverable. However, while this relation can be simplified or replaced, the second issue renders it somewhat moot. While the literature modeling these measures like requirements status is full of various parameters, few if any of these parameters match the firmer works which show how requirements status evolves as a discrete categorization over time across particular review gates and organizationally defined steps. These state-based representations contradict to an extent some of the philosophy underlying the SD approach; however, two examples of real data show more linear trends, perhaps indicating the need for burn down models instead of S-curve models. Overall, for the current scope of P-SEMP, these SD models can likely be skipped in favor of direct representation of task proposals by DES.

The second negative finding followed from the first. Without an appropriate or usable SD model, and more importantly without any clear mapping from the results of the DES to any portion of the SD model, the bi-level simulation was not feasible under the P-SEMP thesis. However, the claims which might be made here are more limited. While in the case of the SD model evidence was shown which point away from that approach, the bi-level simulation cannot be ruled out as firmly. For example, if a team implementing the P-SEMP methodology were to examine some phenomena in addition to the method task proposals which require other models, integrated, bi-level, or otherwise multi-level simulations may become appropriate to investigate once more.

Some of these items will be summarized alongside the formal argument in the concluding chapter. However, as a consequence of these negative and affirmative findings, several changes must be made on the P-SEMP methodology for the proposed Platform version 0.1 to result in the final, as-implemented Platform version 1.0.

12.3 Revised Methodology for P-SEMP

Specifically, the issues which have arisen in the P-SEMP methodology or more specifically the conceptual architecture description of the P-SEMP methodology, are related to the particular SE Analytical Tools specified for the Platform version 0.1.

12.3.1 Multiple Platforms Possible

First of all, it is important to re-iterate that P-SEMP as a methodology is not about only a single platform. Rather, it involves more the application of modeling and simulation to the planning of SE tasks and associated improvements to analysis and decision-making on SE methodology. As such, it is possible if not likely that other integrated sets of analytical tools, thus other variants of the Platform developed across the experiments of the P-SEMP thesis, could be created. Figure 12.1 illustrates the idea in the conceptual architecture description that many variants of the PSEMP Concept Platform.

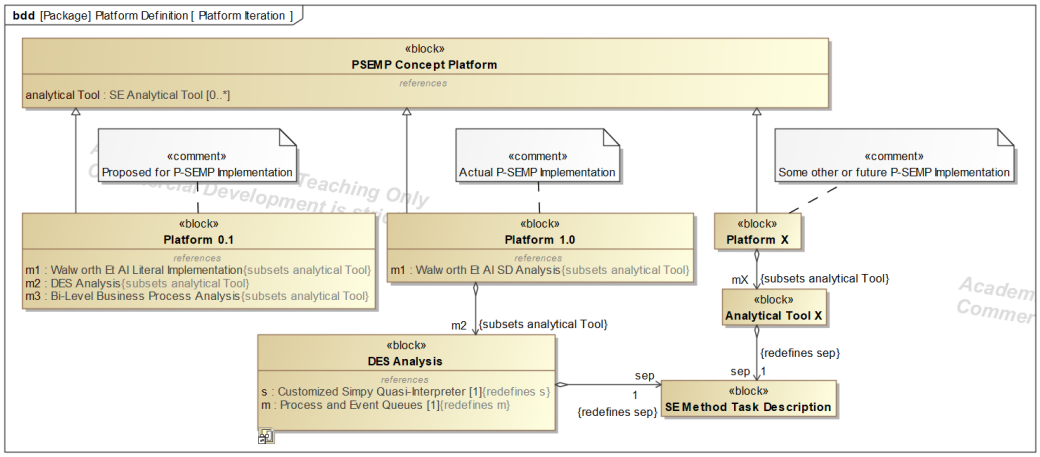


Figure 12.1: Iterative Development of Platforms which are PSEMP Concept Platforms: may include other variations (Platform X) not described in the P-SEMP thesis.

In Figure 12.1, three specialized platforms are shown, specializing the PSEMP Concept Platform. These include the proposed Platform version 0.1, the as-built

Platform 1.0, and a Platform X representative of some potential alternative platform which also may accomplish the objectives of the P-SEMP methodology.

12.3.2 Updates to Methodology vs. Updates to Platform

Chapter 6 described in detail the conceptual platform architecture for and the proposed Platform 0.1 and what it should include. This proposed Platform version 0.1 represented the result which might be obtained had all the proposed experiments of the P-SEMP thesis gone to plan. However, in the P-SEMP methodology, while constructing the Platform according to the specification of Platform 0.1, several issues were encountered per the Affirmative and Negative findings above. These issues were not with the P-SEMP methodology, which ought to be sound according to an engineering interpretation of Zeigler et al[56] for building modeling and simulation environments, as discussed in Chapter 6. However, the issues were in the proposed SE Analytical Tools for the Platform version 0.1. Thus, updates expressed here are the changes which went into creating Platform version 1.0, under the P-SEMP methodology to first construct a platform for systems engineering modeling and planning, and then exercise it.

12.3.3 Updates Leading to Platform Version 1.0

The changes between Platform 0.1 to Platform 1.0 conform to the results from the experiments. The original proposed Walworth et al Analytical Tool (Figure 6.17) was not integrated to the system model in Experiment 3b and therefore is not included in Platform 1.0.

Instead, what is included is the SE Analytical Tool pictured in Figure 12.2. In Figure 12.2, the Python-based tool integration from Experiment 1 and Experiment 3b is described. Furthermore, the integrated result from Experiment 3b is linked to the specified SE Analytical Tool in the conceptual architecture description by

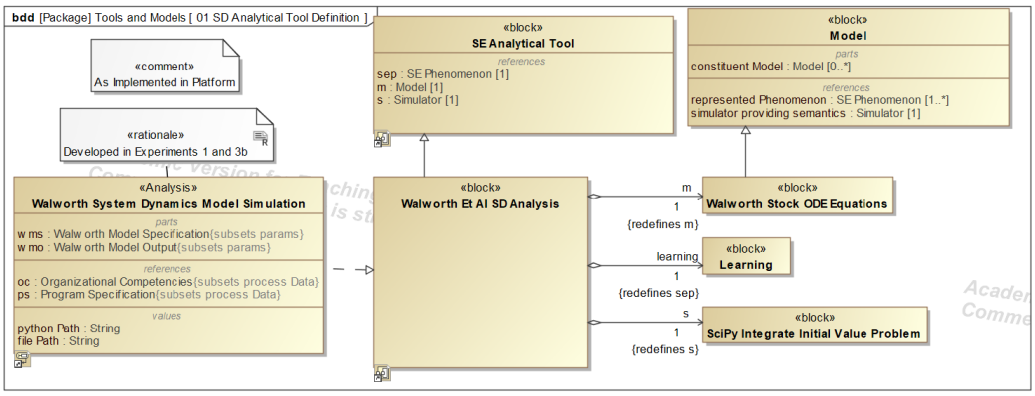


Figure 12.2: Final implementation of the Walworth et al Analytical Tool

a realization edge, providing traceability between the experimental results and the conceptual architecture description. As before in Chapter 6, internals are defined for the SE Analytical Tool in Figure 12.3.

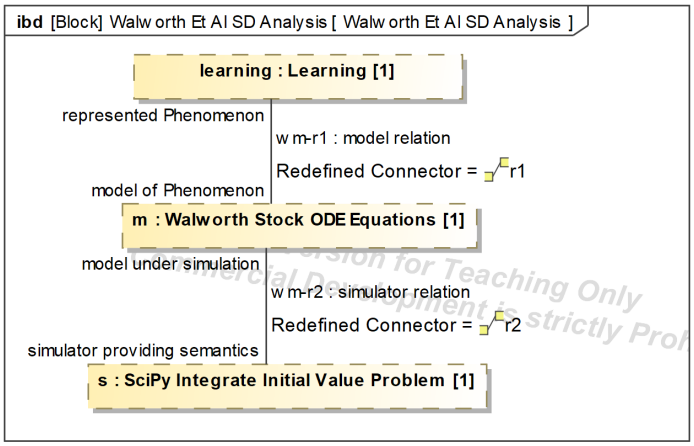


Figure 12.3: Final internals of the Walworth implementation

Figure 12.3 shows that ODEs are used to model Learning and integrated using the Python-based initial value problem solver.

Additionally, beyond the exclusion of the proposed description in Figure 6.17 regarding the Walworth et al model, Platform version 1.0 also excludes the bi-level simulation described by Figure 6.21 according to the Negative findings above. However, as far as Affirmative findings, the proposed DES Analytical Tool is maintained per Figure 6.19 and the items from the experimental results which match to this SE

Analytical Tool and its elements are illustrated by Figure 12.4.

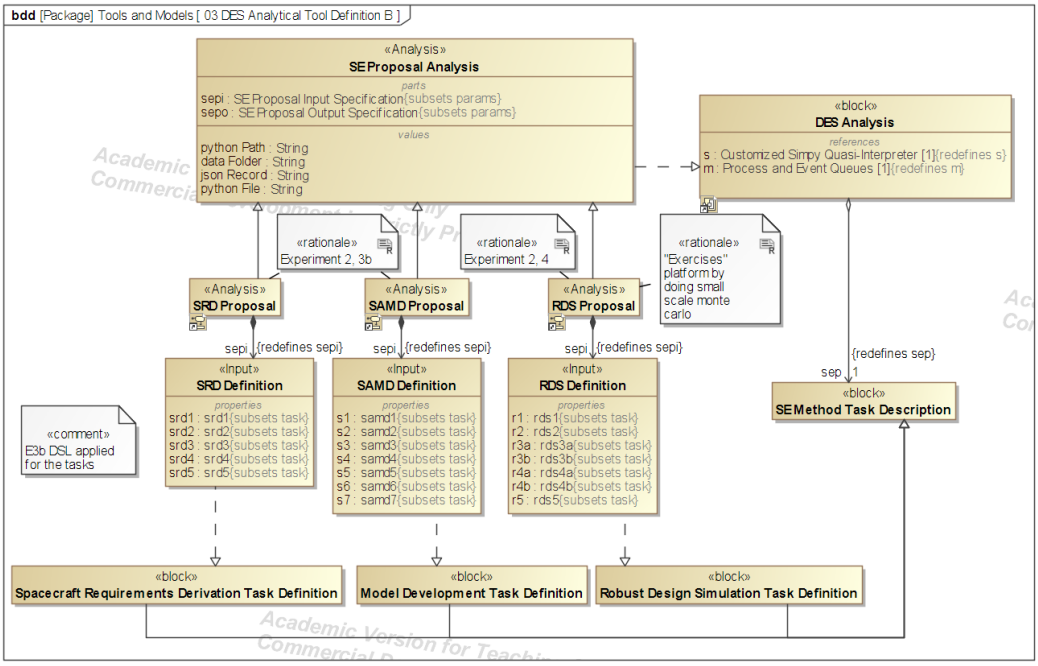


Figure 12.4: Realizations between the results of Experiments 2, 3b, and 4 and the DES Conceptual Architecture elements from Chapter 6.

For the results from Experiment 5, they were illustrated perhaps prematurely in Chapter 6 by Figure 6.12. In Figure 6.12, in the upper right there are the noise variable and DoE content from Experiment 5 which realize conceptual architecture content related to SE methodology tool aspects.

12.3.4 Finalized Platform 1.0

The finalized Platform 1.0 conceptual architecture description is illustrated by Figure 12.5.

Visible in Figure 12.5 are the consequences of the modifications listed above which were the result of the developments between conception of the proposed Platform version 0.1 and the implementation of the experiments. That is, the Walworth model implementation is swapped out, and no bi-level simulation exists. Furthermore, there are the full set of realizing elements available from the experimental results especially

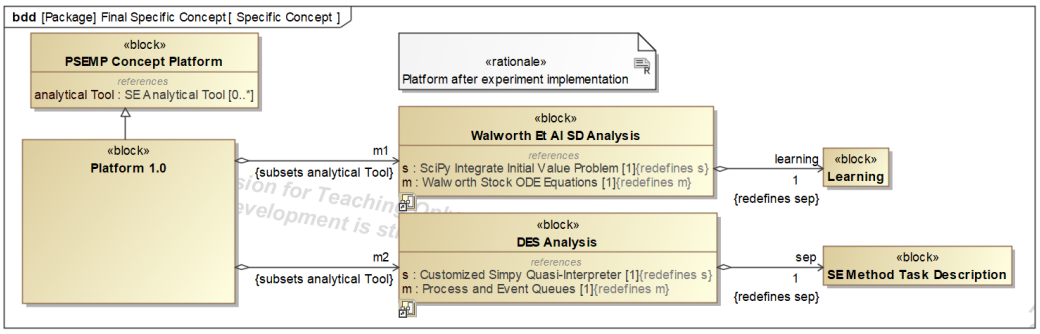


Figure 12.5: The Platform version 1.0 in the conceptual architecture description.

Experiments 3b, 4, and 5 which dealt with specific capabilities added into the system model.

12.3.5 Finalized Trade Study Capability Enabled

Chapter 6 described, in Figure 6.6, the root concept of the P-SEMP Conceptual Architecture description. This included the idea of the PSEMP Concept Platform, with its SE Analytical Tools which of course have been the subject of much discussion in this thesis via the formal argument and the experiments. However, additionally in Figure 6.6 is the idea of the SE Trade Study which uses the tools of the platform to analyze SE methodologies according to some intention or objective of the trade study. With Platform 1.0, the definition of the SE Trade Study can be refined to cover the case of this thesis, especially as described by Experiments 3b and 4, in Figure 12.6.

12.3.6 Concluding the Construction of the Platform

Altogether the establishing of this P-SEMP conceptual architecture description serves primarily to describe what items are involved in the construction of the platform that is the subject of the P-SEMP thesis, as part of the P-SEMP methodology. Specifically, while above and in Chapter 6 the entirety of the Conceptual Architecture Description has been provided, the actual implementation of these concepts remains the content written up by the results and conclusions of the experiments, and which

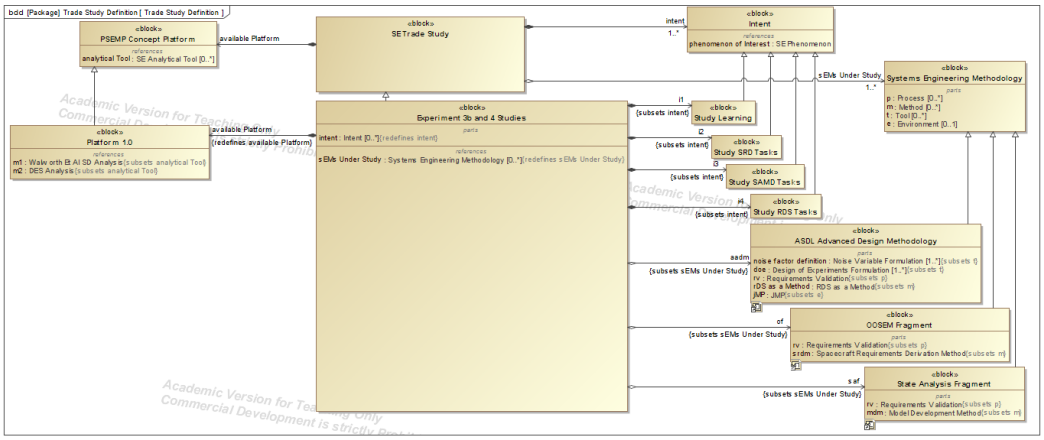


Figure 12.6: Specialized trade study leveraging Platform 1.0 to study SE methodology.

have been linked to the conceptual architecture at appropriate points through realization edges — thereby indicating that the conceptual architecture is specifying what the experimental results realize.

12.3.7 Exercising the Platform

The step of the P-SEMP methodology least described by the conceptual architecture (although implicitly covered by the idea of performing the SE Trade Study) is the step of the P-SEMP methodology to exercise the platform. However, there are various points in Experiments 3b and 4 where the platform is exercised. Arguably, once the results of Experiments 1 and 2 have been integrated to the system model, it is then possible to exploit these analyses from the system model for a variety of purposes. In particular, the example in Experiment 4 is revealing of one possibility: Experiment 4 illustrates a simplistic Monte Carlo simulation of the RDS method it establishes using the system model representation in Figure 10.9 which is the actual implementation of this capability. Figure 10.9 is an activity diagram which will run the DES for the input RDS specification an integer number of times up to the specified quantity for the analysis representation, providing a list of the output values — in this case, duration of the tasks — back to the model. Thus, the tools of the platform are

exercised to study a proposed method.

12.4 Understanding the P-SEMP Methodology by Example

In order to better understand the P-SEMP methodology, several examples are provided to illustrate how the methodology might be applied in different cases. The first case is when studying a new SE method using the existing platform, Platform 1.0. The second case is if Platform 1.0 needs to be augmented or even if a new Platform X must be established.

12.4.1 Example of Studying a New Method with Platform 1.0

First, consider the example of studying a new SE methodology which is proposed to the SE management team. This example will use a popular technique, which neither recognizes the aspects of PMTE nor the ISO 15288 Life Cycle Processes ([6]), according to Kleiner and Kramer 2013[102]. Kleiner and Kramer discuss in their conference paper what they label as a systems engineering process based on using Dassault's tools to construct specific views for Requirements, Functional Design, Logical Design, and Physical Design, and the SE methodology is called RFLP. While Kleiner and Kramer assert a "V model" for the system lifecycle, they do so based on a German standard for mechatronics VDI 2206 and not the standard SE literature[102]. What follows is a description of the procedure for applying the P-SEMP methodology and Platform 1.0 to this paper as a method proposal.

Step 1: Consider the PMTE

The P-SEMP methodology requires structuring information in terms of PMTE — process, methods, tools, and environment. For this purpose, a PMTE table can be created for the RFLP SE methodology proposed by Kleiner and Kramer. For process, standardized processes from [6] and from Honour[2] will be used, specifically as shown

earlier in this thesis for Structured Analysis and for State Analysis. For the current proposal, Figure 12.7 illustrates a take on the PMTE based on Kleiner and Kramer.

Process (Standard)	Methods (Proposed)	Tools	Environment
Requirements Analysis	Create Requirements Model	• Dassault V6+ Environment (ENOVIA and CATIA RFLP Applications) • Word, Excel	Office and IT infrastructure supporting Dassault's platform of tools.
Architectural Definition	Create Functional Design		
	Create Logical Design (Modelica)		
Architectural Definition and/or Design Definition	Create Physical Design (CAD)		

Figure 12.7: PMTE with processes from ISO 15288[6] and methods, tools, environment from Kleiner and Kramer[102].

However, it should be noted that ISO 15288 specifies iteration between architectural definition and design definition for "allocation, partitioning, and alignment of architectural entities to system elements that compose the system"[6]. Therefore, since Kleiner and Kramer are asserting partitions and probably allocation at various steps in their functional design. logical design, and physical design tasks, they must be doing a combination of architectural and design definition across multiple portions of their SE methodology[102]. However, as presented by the authors, the methods proceed in a serial fashion without iteration from requirements coming from stakeholders, to CAD in the physical design. Taking the authors at their word is the simplest approach in this case, for the next step.

Specify Process Nodes for SE Tasks

Using the DSL from Experiment 3b, specify the `Process Node`s definition the proposed tasks of the method. At this point the engineer doing the specification is engaged in *exercising* Platform version 1.0. The result in this case should be similar to other serial methods, like SRD and RDS as pictured in Figure 9.18 and Figure 10.5. In doing so, the engineer must specify the `Process Actionss` for the `Process Node`s. In all likelihood, since there is no branching or iteration defined by Kleiner and Kramer, these actions will be the simpler ones seen in SRD and RDS, incrementing

the passthrough value by a uniform random integer for the first three (Requirements, Functional, Logical) before proceeding to the next (e.g. Figure 9.19), and in the last (Physical) incrementing the passthrough value by a uniform random integer and then `yield`ing zero (e.g. Figure 9.20). The used data for the `Process Node`s should, assuming that the meaning of passthrough value must be duration for this method, choose an initial passthrough value of zero, and apply appropriate minimum and maximum durations for the minimums and maximums in the data so that the uniform random integer addition can properly operate in the simulation.

Specify the Analysis of the SE Tasks

The next step is to create the analysis of the proposed method. A new specialization for the SE Proposal Analysis is created, and the input is defined to subset tasks by the `Process Node`s established in the previous step. Additionally, after deciding whether or not to save the output in the system model, some behavior must be established within the new analysis as shown in the background of Figure 9.30 for an analyses which runs once, or Figure 10.9 for an analysis which will run the simulation some number of times like a simplistic Monte Carlo analysis implementation.

Generate Data

With the analysis and associated driving behavior implemented to run the code from Experiment 2 with the content from the system model, the analysis can now be run to generate data. Should the engineer have chosen not to directly save the results to the system model, the result will be printed to the CST console for the final passthrough value. If the data is saved, then by running the simulation from an instance model the slot value with the output quantities can be preserved as in Figure 10.10. Perhaps the engineer will use their results and the criteria and parameters discussed in Experiment 4, Table 10.4 to make a decision as to how they will proceed. In the conceptual

architecture, at this point the engineer is able to investigate a SE Trade Study for some purpose regarding SE methodology comparison or selection, in particular with respect to SE methods.

Concluding Thoughts on Reuse

The example above regarding the Kleiner and Kramer RFLP illustrates how to apply the P-SEMP methodology and the existing Platform version 1.0 to a new method proposal. In this case, the portion of the P-SEMP methodology which is active is the final phase, exercising the platform. In particular this section has described the procedure for continuing to apply the results of the P-SEMP thesis as-is on further studies similar in nature to the experiments herein.

12.4.2 Example of Augmenting Platform 1.0 or Creating Platform X

However, occasionally phenomena outside those addressed by Platform 1.0's SE Analysis Tools will become relevant to project management and SE planning. In this case, either Platform 1.0 will need to be augmented (resulting in e.g. Platform 1.5) or a new Platform X will need to be created.

Augmenting Platform 1.0

In the example above which exercised Platform 1.0, the assumption was that the methods proposed could be taken on their own without considering the tools or environment in Figure 12.7. However, it should be clear that this is not always valid, especially since Figure 12.7 makes clear that per Kleiner and Kramer, specific tools are intended in each step[102]. For example, the logical design is specified as being in Modelica language. Thus, the tools are not entirely extricable from the methods proposed (regardless of attempts to line up precise definitions of logical vs. functional with respect to the usual standards). One possibility therefore, is that another SE

Analytical Tool might be able to address the ways in which the tools impact the method performance. For example, should the team decide to do dynamic behavior modeling using state machines in SysML instead of differential algebraic equations in Modelica diagrams, how would that impact the duration for creating logical design? Thus, the new phenomenon necessitates the creation of a new SE analytical tool for the platform, and a resulting increment in platform version. Once established, the construction phase of the P-SEMP methodology will conclude again, and then the updated Platform (e.g. version 1.5 for example) could be exercised to analyze the combined effects of tools and method on duration.

Creating a New Platform

Perhaps in a highly regulated environment, not only the SE processes but also the methods (tasks) and even tools/environment are controlled, legislated, or otherwise fixed. However, Martin also shows that for a SE methodology, the technology and Knowledge, Skills, and Abilities (KSAs) of the staff are also important[4]. In this thought experiment, not only would the existing SE Analytical Tools of the platform be potentially irrelevant, but the phenomena available (regarding SE competency) are different. Thus, in discarding the existing SE Analytical Tools and constructing new tools to study for example performance impacts of KSAs, a new Platform X would be constructed. Following the construction of the new platform, the P-SEMP methodology would have the engineers exercise the platform to generate data and do with it as necessary, perhaps in decision-making for training or hiring in this instance. Even in this case, where the proposed Platform 0.1 and the implemented Platform 1.0 are discarded, the root concept (Figure 6.6) remains relevant due to its basis on the fundamentals of modeling and simulation from Zeigler et al[56].

12.4.3 Consequences on the SEMP

There are several aspects of the Systems Engineering Management Plan (SEMP) which might be addressed by the P-SEMP methodology and Platform 1.0. According to Martin, the SEMP must include planning activities and schedules[4, p.81], timelines for reviews and specific methods for SE processes[4, p. 83], as well as the Systems Engineering Master Schedule (defined as "event-based") and the Systems Engineering Detailed Schedule (defined as "calendar-based")[4, pp. 102–103]. Martin also asserts in his guidebook that "iteration will occur between the [SE Management] Planning activity and the [SE Management] Control and Integration activity as the need for replanning arises"[4, p.98]. Creation of SE method models using the DSL established in the P-SEMP thesis, and exercising the method model to determine acceptable minimum and maximum bounds of passthrough values representing method-appropriate quantities, assists in accomplishing these items. In the case of linear or serial methods, the durations of the tasks will be established and directly contribute to schedule specification based on the proposed and/or selected methods for processes whether in initial planning or in later replanning. Iterative method proposals with events defined based on some numerical criteria might also be modeled in Platform 1.0 if the criteria can be treated similarly to state variables for SAMD. These event-driven simulations could help build evidence for or validate a master schedule, perhaps drafted with traditional techniques as described in Experiment 2 other than DES. In any case, the simulation capability inherent in the platform puts the activities of authoring the SEMP on a technical basis. Some SE domains explicitly require the use of simulation capabilities for these purposes, for example in the NASA Schedule Management Handbook which puts forward critical path method[135]; clearly, the proposal by some at NASA for SAMD would require adopting different SE Analytical Tools for study of schedule impacts, as discussed in this thesis.

12.5 Effectiveness of the P-SEMP Methodology Against Benchmarks

In Chapter 6, Section 6.2, various means of comparing or selecting SE methodologies (SEM) were presented from the literature as a form of benchmarking purported SEM Selection Methodologies. This included: Morkevicius et al[101], Gilbertson 2018[100], Honour 2013 (SE-ROI)[2], and of course, the P-SEMP Methodology. There are several key attributes which will be important to consider in selecting a SEM Selection Methodology. First of all, does the SEM Selection Methodology use models? Models enable numerical comparison of the SEM alternatives. Morkevicius[101] does not, while the other three, including P-SEMP, do use models. Further, does the SEM Selection Methodology examine the *methods* of the SEM, which are established as the How per Martin[4] and thus closer to tasks which must be planned for completing the SEM — only P-SEMP addresses this attribute. However, beyond hard numbers, there are also important softer criteria to consider, subjective features or attributes. These subjective features might be weighted in some way during a formalized decision-making exercise, and express important preferences nonetheless. Morkevicius[101] is entirely subjective, while P-SEMP provides some subject parameters for consideration in Experiment 4. Finally, is the SEM Selection Methodology expandable, or can it address new techniques over time without too much effort: Morkevicius[101] is easily expandable in terms of subjective strengths and weaknesses of SEM — which is why it is assumed as the typical default, while P-SEMP is oriented around the expansion of or creation of a platform capable of running models for SE planning purposes. Gilbertson[100] and Honour[2] are both limited in some way; Gilbertson focuses on a predefined set of SE methodological categories, while permitting at each step a novel statistical model or test; Honour uses a statistical model fitted on historical data to calculate the ROI values for various parameters, and this model would need to be re-fit to new data as proposed by Gooden[23] for application to new/existing SEM in

development or developed since Honour's data was collected. The methodologies and some attributes are presented in Table 12.1.

Table 12.1: Comparison of Methodologies for Selecting SE Methodologies.

Attribute	Morkevicius[101]	Gilbertson[100]	Honour[2]	P-SEMP
Models	No	Yes	Yes	Yes
Methods	No	No	No	Yes
Soft	Yes	Yes	No	Yes
Expands	Yes	No	No	Yes

Table 12.1 summarizes the statements of the text above and provides a single, side-by-side view of the commentary established by these SEM Selection Methodologies. It is expected that regardless of any particular Platform version, that the P-SEMP methodology will be applicable to any SE methodological decision-making or analytical effort due to the application of methods in an expandable nature; however, perhaps other more tailored SEM Selection Methodologies addressing other features of SEM with different platforms for analysis might be proposed. As it stands, the Platform version 1.0 resulting from the P-SEMP thesis enables all attributes of the comparison of Table 12.1 and weighs the P-SEMP methodology favorably for selecting SEM.

12.6 Conclusion

The P-SEMP thesis has shown through Affirmative and Negative findings how to create a platform for SE modeling and planning. The platform which resulted from the P-SEMP thesis is recognized as Platform version 1.0 in the P-SEMP conceptual architecture description. Methodologically, the P-SEMP methdology to build (construct) and use (exercise) models according to the P-SEMP Conceptual Architecture has complete logical underpinning per Chapter 6 and has been shown in this current chapter how it might be used again or for new cases. Finally, a comparison of various benchmarking methodologies for selecting SE methodologies. According to all the

capabilities developed across the P-SEMP thesis, the P-SEMP methdology is seen as the currently most-suitable SEM Selection Methodology for modeling and planning new SE methods. However, there are some gaps in particular on Platform version 1.0 as discussed above. For example, phenomena and situations outside the current thesis may require an overhaul of the platform or a new one to be constructed per the discussion on KSAs. However, these activities would still fall under the root concept of the conceptual architecture described earlier in Figure 6.6. What remains at this point, is to summarize and conclude the P-SEMP thesis in terms of its argument by research questions, hypotheses, etc, and to examine avenues for future work.

13

CONCLUSION AND RECOMMENDATIONS

From the beginning, the main objective of this thesis has been to work towards a first attempt at answering how best to do Systems Engineering (SE). That is, more specifically, how to provide a means of simulating differing approaches for the validation process in SE early in the lifecycle process (different methods which perform requirements validation), while also provided tools to assist some such validation tasks in feasibility estimation. Martin has put forward the structure of SE Methodology (SEM). According to Martin, SEM consists of process, methods, tools, and environment surrounded by technological capabilities and limitations, as well as personnel and their knowledge, skills, and abilities[4]. If the tools, environment, knowledge, skills, abilities, and technology are fixed, then the performance of SE processes should depend on the tasks described by the SE methods. Thus there is an opportunity to improve SE process performance by tuning SE methods, according to measures and models in the SE Management Plan (SEMP). In response, the Platform for Systems Engineering Modeling and Planning (P-SEMP) has been proposed and defended in this document.

13.1 Recap of the Hypotheses

The hypotheses presented in this document included:

Hypothesis 1 If a model for a SE measure predicts progress in a SE process in correlation with a lagging indicator such as cost, schedule (often combined as effort), etc, then the model provides a basis for decision making on the method for a SE process.

Hypothesis 1a If requirements status is parameterized, then important parameters can be identified and understood.

Hypothesis 2 If SE methods are also processes per Martin[4], then they can be simulated by process models like DES or ABS.

Hypothesis 3 If SE methods are treated like other business processes, then bi-level hybrid simulation will provide better parameterization for task-planning purposes.

Hypothesis 4 If a labeling for probabilistic methods is applied to a customization of an MBSE language with transformation via intermediary representation to target analytical environments, then confidence will be increased in the system representation of the resulting analytical models.

Experiment 1 was targeted at Hypothesis 1a, but found it to be probably rejected. Notably, later parameterization discussion across Experiment 2, 3, and 4 show that the primary issues in Hypothesis 1a is requirements status; parameterizing the SEM in terms of the task architecture showed more promise due to the issues experienced with the Walworth model and discussed in the Grenn model during Experiment 1.

Experiment 2 resulted in the acceptance of Hypothesis 2 by showing the utility of the event-based modeling paradigm in representing SEM tasks through DES.

Hypothesis 3 was rejected due to the incompatibilities between the models used.

Hypothesis 1 was accepted as a result of the parameterization developed under Experiment 3b and Experiment 4 for exploring the SEM task architectures.

Hypothesis 4 was accepted due to the results of Experiment 5 in formulating DoE generation capability and noise variable representation with probability distributions and monte carlo simulation via a system model.

13.2 Recap of the Research Questions

Experiments 1-4 demonstrated the construction and utilization of analytical models for formulating and comparing SE task proposals, as well as additional criteria to consider in making decisions between them. Limits and performance objectives can be set according to the task architecture method model inputs and outputs. However, leading indicator models were less successful. In the development of the integrations of Experiment 3, a new research question Research Question 3a was established. In the results here, the parameterization resulting from the task architecture was preferred. To culminate in the development of these techniques, a domain-specific language was formulated to assist in posing method models as plans in a system model, such that they can be fed directly into a simulator for performance analysis. Finally, to help ensure that analytical models of system performance are also correct, a set of Factor Representations, DoE Specification, and Noise Variable Distributions with mappings to intermediate representations were implemented to enable facets of robust design simulation.

13.3 Recap of the P-SEMP Methodology

The P-SEMP thesis was conducted according to its formal argument; however, on product of the P-SEMP thesis is the P-SEMP methodology and associated conceptual architecture. For these products, there are several conclusions and recommendations to note.

13.3.1 Recommendations in terms of the Conceptual Architecture

The P-SEMP methodology consists of constructing and exercising a platform for SE modeling and planning. The P-SEMP conceptual architecture in Figure 6.6, Figure 12.5, and Figure 12.6 illustrate the final pieces of the P-SEMP thesis at the conceptual

level towards a version 1.0 platform for the purposes of conducting SE Trade Studies on SEM. Chapter 12 provided additional detail as far as what might happen if an engineer would like to apply the existing Platform 1.0, or if a new platform or revised platform might become necessary. The new or revised platform may become necessary if applying the P-SEMP methodology for phenomena outside those currently listed in the conceptual architecture for the included SE Analytical Tools, such as Knowledge, Skills, and Abilities for SE capability of an organization or individual team members. Chapter 12 provided a detailed discussion of these aspects, and this section of the conclusion summarizes the main points regarding methodology from that Chapter. These issues discussed in the previous chapter would result in a recommendation to construct a Platform X or Z in the Conceptual Architecture to focus on new phenomena with new models and simulators per Figure 12.1.

13.3.2 What the P-SEMP Methodology Can and Cannot Do

The P-SEMP methodology can guide the process of establishing and using models to make decisions on the content of SEM. The specific contribution of the P-SEMP thesis is the application of the methodology towards the phenomena of methods, specifically method proposals or proposed task descriptions, in the posing of new SEM for consideration when authoring a SEMP. More limitations exist for particular platforms, specifically Platform 1.0 established by the P-SEMP thesis over the course of the experiments. Platform 1.0 cannot offer analysis on tools, environment, KSAs, or technology; it is restricted to analyzing methods as task proposals via DES and perhaps learning through the simplified Walworth ODEs. These other items, visible from Figure 1.2, remain important in SE planning. As discussed in Chapter 8, Bott and Mesmer[116] have investigated engineering cognition models for completion of tasks by individual engineers. The behavior and competency of individual engineers is important in project staffing and can accelerate or diminish task performance as

a variable outside the scope of the P-SEMP thesis, and as discussed in a wide range of literature[136, 137, 138, 139, 140, 141]. Future work could investigate the impacts both of planned tasks and planned team capability on SE process outcomes to find the best combination. Along the different categories outlined by Figure 1.2, such future work might include studying the effects of modifying available tools for the SE methodology, or technologies which support the SE methodology, or other features of SE methodology related to system design. These possibilities include in part:

- Choice of language. The paradigm of MBSE necessitates the selection of a language, and this thesis has used SysML heavily in its implementation. However, future work may investigate the effect of different languages such as AADL, OPL, others, or combinations on task performance and SE methdology selection.

- System of Interest dependency. SE methodology is usually tailored to the System of Interest. As discussed with Gilbertson[100] there is the Class of System Problem, but also various stakeholder concerns on physics, disciplines, or fields of study may affect the features of appropriate SE methodology and drive selection of one SE methodology versus another. Optimal SE methods for healthcare systems as in Gilbertson may be different than for aircraft, and likewise different from the canonical system FireSat used in the P-SEMP thesis.

- As fast as possible. This thesis inferred impacts on cost via direct or indirect measure of schedule performance through Experiment 2 and Experiment 4 with the view that cost would be the ultimate measure of SE performance in deducing the effect of SE methodology on organizational bottom-line. However, as Honour[2] discusses, other metrics are important. Management may prefer to minimize the duration at any cost, given that the effort succeeds. Future task planning capability may apply architecture optimization techniques as dis-

cussed in Experiment 4 under this formulation of objective function to search a space of SE methodology — perhaps by selecting between a finite number of SE methodologies or by constructing new proposals of SE methods/tasks, thereby generating alternatives to be evaluated against the schedule criterion.

- Future-proofing. Management may be concerned with the longevity of tools and environments acquired for the PMTE of particular SE methodology, and whether the SE methodology will remain relevant over the development of multiple products across a timeline of months or years. Selection of SE methodology under these organizational constraints poses a different problem for future consideration.

Despite the restrictions, however, the P-SEMP thesis is still an improvement per Table 12.1 whereby other methodologies for selecting or comparing SEM lack the desired capability. Proposed items which went particularly well include the DSL which assists in realizing the P-SEMP conceptual architecture SE Analytical Tool for analyzing method proposals described in Figure 6.19 and Figure 12.4. As far as what was validated, especially in terms of the proposed Platform 0.1, these descriptions of the SE Analytical Tool for direct simulation of the method proposals was validated as the best approach thus far, while the SD model was mostly ruled out, at least insofar as the parameters associated with learning appear to be indicative of requirements status.

13.4 General Recommendations

A few recommendations can be made on the basis of the experimental results. For example, it is clear that the Walworth model is not suitable. However, there could be some other SD model which performs better, and it has not been completely ruled out. Further, instead of the learning curve archetype, the question arises whether a

model for a burn down might provide a similarly compelling narrative in the realm of SD. Additionally, measures outside of requirements relating to architecture may still be more applicable in terms of learning curve, and the application of models like Walworth to those measures may provide valuable insights and extend the work discussed in this thesis. On the subject of DES, a clear avenue to improve would be a full parser implementation, potentially allowing improvements in parameterization and therefore comparability of the method models. Finally, the possibility for systematic architecture exploration using optimization routines on the proposed task architectures may be promising, especially if combined with cost models and some calibration data for how much time or effort particular tasks may take based on various SEM factors (tools, environment, etc).

13.5 Practical Benefits

This thesis discovered key issues in the formulation of leading indicator models. It proposed a formulation of a domain-specific language for simulating method models by discrete-events. It provided a parameterization and comparison of three particular method models: spacecraft requirements derivation, state analysis model development, and robust design simulation. Finally, implementations of probability and DoE suitable for assisting robust design simulation were implemented on a spacecraft link analysis example. Overall, the core benefit is the ability to obtain a probabilistic performance assessment on task proposals for systems engineering methods described within a system model.

Appendices

APPENDIX A
PYTHON CODE SUPPORTING P-SEMP

In order to run the experiments, several key codes were built. These codes will be outlined in this appendix.

A.1 In Support of the Walworth Model

Experiment 1 sought to build the model in Walworth et al[42]. In the end, an approach leveraging ordinary differential equations for an initial value problem was applied to the system of equations described by the assumed/simplified model, as discussed in Experiment 1. A version of the finalized code is in Listing A.1. This is a standalone version of the model.

Listing A.1: Base implementation of the Walworth model in Python.

```
1  import json
2  import numpy as np
3  import scipy.integrate as spi
4  import matplotlib.pyplot as plt
5  import cProfile
6  import pstats
7  from pstats import SortKey
8  from time import thread_time
9  from multiprocessing import Pool
10 from functools import partial
11 from copy import deepcopy
12
13 DEFAULTS = {
14         "intensity": 0.,
15         "learning_power": 1.,
```

```
16          "number_of_staff": 2.,
17          "total_number_of_tasks": 100.,
18          "effort": 0.5,
19          "quality_of_work_done": 0.8,
20          "urgency": 5.,
21          "attention_span": 0.,
22          "discovery_factor": 0.5
23          }
24
25  class WalworthModel():
26
27      def __init__(self, **kwargs):
28          self.__dict__.update(DEFAULTS)
29          if kwargs is not None:
30              self.__dict__.update(kwargs)
31          assert "intensity" in self.__dict__.keys()
32          assert "learning_power" in self.__dict__.keys()
33          assert "number_of_staff" in self.__dict__.keys()
34          assert "total_number_of_tasks" in self.__dict__.keys()
35          assert "effort" in self.__dict__.keys()
36          assert "quality_of_work_done" in self.__dict__.keys()
37          assert "attention_span" in self.__dict__.keys()
38          assert "discovery_factor" in self.__dict__.keys()
39          self.work_to_be_done = self.total_number_of_tasks# Stock
40          self.work_really_done = 0.       # Stock
41          self.known_rework = 0.           # Stock
42          self.undiscovered_rework = 0.    # Stock
43          self.y = [self.work_to_be_done, self.work_really_done,
44                  self.known_rework, self.undiscovered_rework]
45          self.ynames = ["work_tbd", "work␣done", "rescheduled", "
                undiscovered␣rework"]
46          self.learning_about_the_problem = 0.     # Intermediate
47          self.learning_potential_of_team = 0.     # Intermediate
```

```
48          self.learning_rate = 0.                 # Intermediate
49          self.work_done_wrong_rate = 0.          # Intermediate
50          self.work_done_right_rate = 0.          # Intermediate
51          self.reschedule_rate = 0.               # Intermediate
52          self.rework_discovery_rate = 0.         # Intermediate
53          self.res = None
54          if self.yinit[0] != self.total_number_of_tasks:
55              self.yinit[0] = self.total_number_of_tasks
56
57      def _calc_problem_learning(self):
58
59          return self.work_to_be_done * self.effort *\
60                  self.work_really_done / self.total_number_of_tasks
61
62      def _calc_learning_potential(self):
63
64          return self.number_of_staff * self.learning_power *\
65                  self.work_to_be_done # Learning Potential of the Team
66
67      def update_intermediates(self):
68
69          self.work_to_be_done = self.y[0]
70          self.work_really_done = self.y[1]
71          self.known_rework = self.y[2]
72          self.undiscovered_rework = self.y[3]
73          self.learning_about_the_problem = self._calc_problem_learning()
74          self.learning_potential_of_team = self._calc_learning_potential
                ()
75          self.learning_rate = self.learning_about_the_problem +\
76                  self.learning_potential_of_team
77          self.work_done_wrong_rate = (1. - self.quality_of_work_done) *\
78                  self.learning_rate
79          self.work_done_right_rate = self.quality_of_work_done *\
```

```
80                  self.learning_rate
81          self.reschedule_rate = self.known_rework * self.urgency
82          self.rework_discovery_rate = self.discovery_factor * self.
                undiscovered_rework
83
84          return 0
85
86      def dynamics(self, t, y):
87          """
88          definition of sd model rate calculations
89          """
90          self.y = y
91          q = self.update_intermediates()
92          rate_work_be_done = self.reschedule_rate - self.learning_rate
93          rate_work_real_done = self.work_done_right_rate
94          rate_known_rework = self.rework_discovery_rate - self.
                reschedule_rate
95          rate_undiscovered = self.work_done_wrong_rate -\
96                  self.rework_discovery_rate
97          dydt = [rate_work_be_done, rate_work_real_done,\
98                  rate_known_rework, rate_undiscovered]
99          return dydt
100
101     def run(self):
102         """
103         run the sd model
104         """
105         self.res = spi.solve_ivp(lambda t, y: self.dynamics(t, y),\
106                 [self.tspan[0], self.tspan[-1]], self.yinit, t_eval=self
                    .tspan)
107
108     def duke_plotter(self, times, states, fignum):
109         """
```

```
110         See citation 107, Duke, Python ODEs
111         https://pundit.pratt.duke.edu/wiki/Python:
                Ordinary_Differential_Equations/Examples
112         """
113
114
115
116 def main():
117     data = np.load('designtest.npz')
118     data_shape = data['cases'].shape
119     run_data = []
120     tspan = np.linspace(0, 8, 100)
121     yinit = [100, 0, 0, 0]
122     for x in range(data_shape[0]):
123         data_vec = data['cases'][x,:].tolist()
124         run_data.append(WalworthModel(intensity=data_vec[0],\
125                 learning_power=data_vec[1],\
126                 number_of_staff=data_vec[2],\
127                 total_number_of_tasks=data_vec[3],\
128                 effort=data_vec[4],\
129                 quality_of_work_done=data_vec[5],\
130                 urgency=data_vec[6],\
131                 attention_span=data_vec[7],\
132                 discovery_factor=data_vec[8],\
133                 yinit=yinit.copy(),tspan=tspan.copy()))
134     for wm in run_data:
135         wm.run()
136     np.savez_compressed("out.npy", result=run_data, allow_pickle=True)
137
138
139 if __name__=="__main__":
140     cProfile.run('main()', 'runstat')
141     p = pstats.Stats('runstat')
```

```
142     p.sort_stats(SortKey.TIME).print_stats(20)
```

In order to run a design of experiments on Walworth, a DoE generator was created. Listing A.2 presents the source code.

Listing A.2: Walworth DoE Generator

```
1  from pyDOE2 import *
2
3
4  DEFAULTS_DOE = {
5          "intensity": {"min":0.,"max":10.},
6          "learning_power": {"min":0.,"max":2.},
7          "number_of_staff": {"min":1.,"max":10.},
8          "total_number_of_tasks": {"min":50.,"max":200.},
9          "effort": {"min":0.01,"max":2.0},
10         "quality_of_work_done": {"min":0.,"max":1.},
11         "urgency": {"min":0.,"max":10.},
12         "attention_span": {"min":0.,"max":10.},
13         "discovery_factor": {"min":0.,"max":100.}
14         }
15
16 class DesignFactor:
17
18     def __init__(self,**kwargs):
19         self.__dict__.update(kwargs)
20         assert "imin" in self.__dict__.keys()
21         assert "imax" in self.__dict__.keys()
22         assert "name" in self.__dict__.keys()
23
24     def __repr__(self):
25         return "var::{}(min={},max={})".format(self.name, self.imin,
               self.imax)
26
```

```
27      def normalize(self, x):
28          diff = self.imax - self.imin
29          return (2./diff)*(x - self.imin) - 1.
30
31      def denormalize(self, xn):
32          diff = self.imax - self.imin
33          return (diff/2.)*(xn + 1.) + self.imin
34
35
36  t1 = bbdesign(9, center=1)
37
38  vlist = []
39  for k,v in DEFAULTS_DOE.items():
40      for j,w in v.items():
41          if j == "min":
42              this_imin = w
43          if j == "max":
44              this_imax = w
45      vlist.append(DesignFactor(name=k,imin=this_imin,imax=this_imax))
46
47  shape_t1 = t1.shape
48  outs = np.zeros(shape_t1)
49  for x in range(shape_t1[1]):
50      outs[:,x] = vlist[x].denormalize(t1[:,x])
51
52  np.savez_compressed('designtest',cases=outs)
```

Listing A.3 was built to read and plot data for Experiment 1.

Listing A.3: Reader for Walworth Data

```
1  import numpy as np
2  from Walworth_ODE_2 import WalworthModel
3  import scipy.integrate as spi
```

```
4  import matplotlib.pyplot as plt
5  from mpl_toolkits.mplot3d import axes3d
6
7  def plot3Dfun(datalists):
8      fig = plt.figure()
9      ax = fig.gca(projection='3d')
10     X = datalists["result"][0].res.t
11     Z = range(len(datalists["result"]))
12     X, Z = np.meshgrid(X, Z)
13     YY = [datalists["result"][k].res.y[0] for k in range(len(Z))]
14     print(len(YY))
15     print(len(YY[0]))
16     Y = np.zeros(Z.shape)
17     print(X.shape)
18     print(Z.shape)
19     print(Y.shape)
20     Y = np.asarray(YY)
21     ax.plot_surface(X, Y, Z, cmap='viridis')
22     plt.show()
23
24
25 thedata = np.load("out.npy.npz", allow_pickle=True)
26 print([len(x.res.t) for x in thedata["result"]])
27
28 thedata["result"][50].duke_plotter(thedata["result"][50].res.t, thedata[
       "result"][50].res.y, 1)
29
30 print(thedata["result"][100].__dict__.items())
31
32 plot3Dfun(thedata)
```

Finally, Listing A.4 is a modified version of the Walworth model was created to support command line evaluation. This version is for integration with the system

model. This script was discussed in part during Experiment 3.

Listing A.4: Command Line Walworth Model

```
1  import json
2  import argparse
3  import numpy as np
4  import scipy.integrate as spi
5  import matplotlib.pyplot as plt
6  import cProfile
7  import sys
8  from copy import deepcopy
9
10 DEFAULTS = {
11          "intensity": 0.,
12          "learning_power": 1.,
13          "number_of_staff": 2.,
14          "total_number_of_tasks": 100.,
15          "effort": 0.5,
16          "quality_of_work_done": 0.8,
17          "urgency": 5.,
18          "attention_span": 0.,
19          "discovery_factor": 0.5
20          }
21
22 DEFAULTS_ARRAYS = {"tspan": np.linspace(0, 8, 100), "yinit":[100, 0, 0,
       0]}
23
24 DEFAULTS_COMBINED = dict()
25 DEFAULTS_COMBINED.update(DEFAULTS)
26 DEFAULTS_COMBINED.update(DEFAULTS_ARRAYS)
27
28
29 class WalworthModel():
```

```
30
31      def __init__(self, **kwargs):
32          self.__dict__.update(DEFAULTS)
33          if kwargs is not None:
34              self.__dict__.update(kwargs)
35          assert "intensity" in self.__dict__.keys()
36          assert "learning_power" in self.__dict__.keys()
37          assert "number_of_staff" in self.__dict__.keys()
38          assert "total_number_of_tasks" in self.__dict__.keys()
39          assert "effort" in self.__dict__.keys()
40          assert "quality_of_work_done" in self.__dict__.keys()
41          assert "attention_span" in self.__dict__.keys()
42          assert "discovery_factor" in self.__dict__.keys()
43          self.work_to_be_done = self.total_number_of_tasks# Stock
44          self.work_really_done = 0.       # Stock
45          self.known_rework = 0.           # Stock
46          self.undiscovered_rework = 0.    # Stock
47          self.y = [self.work_to_be_done, self.work_really_done,
48                  self.known_rework, self.undiscovered_rework]
49          self.ynames = ["work_tbd", "work␣done", "rescheduled", "
               undiscovered␣rework"]
50          self.learning_about_the_problem = 0.     # Intermediate
51          self.learning_potential_of_team = 0.     # Intermediate
52          self.learning_rate = 0.                  # Intermediate
53          self.work_done_wrong_rate = 0.           # Intermediate
54          self.work_done_right_rate = 0.           # Intermediate
55          self.reschedule_rate = 0.                # Intermediate
56          self.rework_discovery_rate = 0.          # Intermediate
57          self.res = None
58          if self.yinit[0] != self.total_number_of_tasks:
59              self.yinit[0] = self.total_number_of_tasks
60
61      def _calc_problem_learning(self):
```

```
62
63          return self.work_to_be_done * self.effort *\
64                  self.work_really_done / self.total_number_of_tasks
65
66      def _calc_learning_potential(self):
67
68          return self.number_of_staff * self.learning_power *\
69                  self.work_to_be_done # Learning Potential of the Team
70
71      def update_intermediates(self):
72
73          self.work_to_be_done = self.y[0]
74          self.work_really_done = self.y[1]
75          self.known_rework = self.y[2]
76          self.undiscovered_rework = self.y[3]
77          self.learning_about_the_problem = self._calc_problem_learning()
78          self.learning_potential_of_team = self._calc_learning_potential
                ()
79          self.learning_rate = self.learning_about_the_problem +\
80                  self.learning_potential_of_team
81          self.work_done_wrong_rate = (1. - self.quality_of_work_done) *\
82                  self.learning_rate
83          self.work_done_right_rate = self.quality_of_work_done *\
84                  self.learning_rate
85          self.reschedule_rate = self.known_rework * self.urgency
86          self.rework_discovery_rate = self.discovery_factor * self.
                undiscovered_rework
87
88          return 0
89
90      def dynamics(self, t, y):
91          """
92          definition of sd model rate calculations
```

```
93          """
94          self.y = y
95          q = self.update_intermediates()
96          rate_work_be_done = self.reschedule_rate - self.learning_rate
97          rate_work_real_done = self.work_done_right_rate
98          rate_known_rework = self.rework_discovery_rate - self.
                reschedule_rate
99          rate_undiscovered = self.work_done_wrong_rate -\
100                 self.rework_discovery_rate
101         dydt = [rate_work_be_done, rate_work_real_done,\
102                 rate_known_rework, rate_undiscovered]
103         return dydt
104
105     def run(self):
106         """
107         run the sd model
108         """
109         self.res = spi.solve_ivp(lambda t, y: self.dynamics(t, y),\
110                 [self.tspan[0], self.tspan[-1]], self.yinit, t_eval=self
                    .tspan)
111
112     def duke_plotter(self, times, states, fignum):
113         """
114         See citation 107, Duke, Python ODEs
115         https://pundit.pratt.duke.edu/wiki/Python:
                Ordinary_Differential_Equations/Examples
116         """
117
118
119 def main():
120     data = np.load('designtest.npz')
121     data_shape = data['cases'].shape
122     run_data = []
```

```
123     tspan = np.linspace(0, 8, 100)
124     yinit = [100, 0, 0, 0]
125     for x in range(data_shape[0]):
126         data_vec = data['cases'][x,:].tolist()
127         run_data.append(WalworthModel(intensity=data_vec[0],\
128                 learning_power=data_vec[1],\
129                 number_of_staff=data_vec[2],\
130                 total_number_of_tasks=data_vec[3],\
131                 effort=data_vec[4],\
132                 quality_of_work_done=data_vec[5],\
133                 urgency=data_vec[6],\
134                 attention_span=data_vec[7],\
135                 discovery_factor=data_vec[8],\
136                 yinit=yinit.copy(),tspan=tspan.copy()))
137     for wm in run_data:
138         wm.run()
139     np.savez_compressed("out.npy", result=run_data, allow_pickle=True)
140
141
142
143 parser = argparse.ArgumentParser("WalworthODE3")
144 parser.add_argument("--inputdata", type=str, default='')
145 args = parser.parse_args()
146 if args.inputdata == '':
147     cProfile.run('main()')
148 else:
149     inputDictionary = json.loads(args.inputdata)
150     runDictionary = dict()
151     for k, v in inputDictionary.items():
152         newkey = k.replace(' ', '_')
153         if newkey in DEFAULTS_COMBINED.keys():
154             if newkey == 'tspan':
155                 runDictionary[newkey] = np.linspace(v[0], v[1], int(v
```

```
                    [2]))
156             else:
157                 runDictionary[newkey] = v
158     for k, v in DEFAULTS_COMBINED.items():
159         if k != runDictionary.keys():
160             runDictionary[k] = v
161     if runDictionary["total_number_of_tasks"] != runDictionary["yinit"
            ][0]:
162         runDictionary["yinit"][0] = runDictionary["total_number_of_tasks
                "]
163     wm = WalworthModel(
164         intensity=runDictionary["intensity"],
165         learning_power=runDictionary["learning_power"],
166         number_of_staff=runDictionary["number_of_staff"],
167         total_number_of_tasks=runDictionary["total_number_of_tasks"],
168         effort=runDictionary["effort"],
169         quality_of_work_done=runDictionary["quality_of_work_done"],
170         urgency=runDictionary["urgency"],
171         attention_span=runDictionary["attention_span"],
172         discovery_factor=runDictionary["discovery_factor"],
173         yinit=runDictionary["yinit"].copy(),
174         tspan=runDictionary["tspan"].copy()
175     )
176     wm.run()
177     output = dict().fromkeys(['time', 'work to be done', 'work really
            done', 'undiscovered rework', 'known rework'])
178     output["time"] = wm.res.t.tolist()
179     output["work to be done"] = wm.res.y[0].tolist()
180     output["work really done"] = wm.res.y[1].tolist()
181     output["undiscovered rework"] = wm.res.y[2].tolist()
182     output["known rework"] = wm.res.y[3].tolist()
183     sys.stdout.write(json.dumps(output))
```

Note that in both Listing A.1 and Listing A.4, the code block from [109] has been removed. This code block would be necessary to run Listing A.3.

A.2 In Support of DES

Experiment 2 required several codes to assist in the discrete event simulation (DES). First of these is Listing A.5 for data analysis, looking at the interarrival times of various simulations and project data.

Listing A.5: Error Data Analysis and Revision Interarrival

```
1  import csv
2  import numpy as np
3  from scipy.stats import expon
4  import matplotlib.pyplot as plt
5
6  FNAMES = ['20200919samples-1.csv', '20200919samples-2.csv', '20200919
       samples-3.csv']
7  STUDYNAMES = [('Exponential Error Detection Fit', 'Simulation'), ('
       Requirement Revision', 'Recorded'),
8                 ('Tuned Error Detection', 'Simulation')]
9
10 def readData(fname):
11     # read in data
12     with open(fname, 'r', encoding='utf-8-sig') as csvfile:
13         dreader = csv.reader(csvfile)
14         header = next(dreader, None)
15         col = {}
16         for h in header:
17             col[h] = []
18         for row in dreader:
19             for h, v in zip(header, row):
20                 col[h].append(v)
```

```
21      return col, header
22
23  # process fit
24
25  for aname, study in zip(FNAMES, STUDYNAMES):
26      col, header = readData(aname)
27      newdata = [float(v) for v in col[header[-1]]]
28      loc, scale = expon.fit(np.asarray(newdata))
29      print(loc)
30      print(scale)
31      mean, var, skew, kurt = expon.stats(loc=loc, scale=scale, moments='
            mvsk')
32      print('Mean: {}, Variance: {}, skew: {}, kurtosis: {}'.format(mean,
            var,skew,kurt))
33      x = list(np.random.exponential(scale=scale, size=1000))
34      y = np.linspace(expon.ppf(0.001, loc=loc, scale=scale),
35                      expon.ppf(0.999, loc=loc, scale=scale), 10000)
36      fig, ax = plt.subplots(1, 1)
37      ax.plot(y, expon.pdf(y, loc=loc, scale=scale),
38              'k-.', lw=5, alpha=0.6, label='{} Fit PDF'.format(study[1]))
39      ax.hist(x, density=True, histtype='stepfilled', alpha=0.3, label='
            Fitted PDF Sample')
40      ax.hist([round(x) for x in newdata], density=True, histtype='step',
            lw=2, label='{} Data'.format(study[1]))
41      ax.legend(loc='best', frameon=False)
42      ax.set_title('{} Fit'.format(study[0]))
43      plt.xlabel('Hours')
44      plt.show()
```

Two draft codes provided a simple implementation of OOSEM-Lite SRD and SA-MD. Listing A.6 is the simple OOSEM SRD implementation. Listing A.7 is the simple SA-MD implementation.

Listing A.6: Simple SRD DES Simulation

```
"""
Test for OOSEM Spacecraft Requirements Definition Simulation
"""
import simpy

class SRD:
    def __init__(self, env):
        self.env = env
        self.finished = env.event()
        self.p1_proc = env.process(self.p1(env))

    def p1(self, env):
        print('1.  Defining external interfaces for mission elements...'
            )
        yield env.process(self.p2(env))

    def p2(self, env):
        print('2.  Specifying spacecraft behavior supporting required
            functions and states')
        yield env.process(self.p3(env))

    def p3(self, env):
        print('3.  Identifying potential failure modes')
        yield self.env.process(self.p4(env))

    def p4(self, env):
        print('4.  Specifying spacecraft performance, physical, and
            quality characteristics')
        yield env.process(self.p5(env))

    def p5(self, env):
```

```
29          print('5.  Capture system requirements in terms of a black box
                specification to refine text-based requirements')
30          self.finished.succeed()
31          yield env.timeout(0)
32
33  env = simpy.Environment()
34  print("creating environment for:\n::OOSEM-FireSat System Requirements
        Definition::")
35  srd = SRD(env)
36  print("srd created, running sim...")
37  env.run()
38  print('end sim')
```

Listing A.7: Simple SAMD DES Simulation

```
1   """
2   Test for SA Model Development Simulation
3   """
4   import simpy
5   import random
6
7
8   class SMD:
9       def __init__(self, env, sv=10):
10          self.env = env
11          self.new_state_variables = sv
12          self.sv_hist = [("init", sv)]
13          self.p1_proc = env.process(self.p1(env))
14          self.iters = 0
15
16      def p1(self, env):
17          print('1.  Identify Needs')
18          yield env.process(self.p2(env))
```

```
19
20      def p2(self, env):
21          print('2.  Identify State Variables')
22          self.new_state_variables = self.new_state_variables + int(random
                .uniform(0, 2))
23          self.sv_hist.append(("Step 2", self.new_state_variables))
24          yield env.process(self.p3(env))
25
26      def p3(self, env):
27          print('3.  Define State Models for State Variables')
28          self.new_state_variables = 0
29          print('All state variables consumed - modeled')
30          self.sv_hist.append(("Step 3", self.new_state_variables))
31          yield env.process(self.p4(env))
32
33      def p4(self, env):
34          print('4.  Identify Measures for State Estimation')
35          yield env.process(self.p5(env))
36
37      def p5(self, env):
38          print('5.  Define Measurement Models')
39          test = random.random()
40          if round(test)==1.0:
41              print('Step 5 discovered state variables!')
42              self.new_state_variables = self.new_state_variables + int(
                    random.uniform(1, 5))
43              self.sv_hist.append(("Step 5", self.new_state_variables))
44          yield env.process(self.p6(env))
45
46      def p6(self, env):
47          print('6.  Identify commands to control state variables')
48          yield env.process(self.p7(env))
49
```

```
50      def p7(self, env):
51          print('7.  Define command models')
52          test = random.random()
53          if round(test) == 1.0:
54              print('Step 7 discovered state variables!')
55              self.new_state_variables = self.new_state_variables + int(
                    random.uniform(1, 5))
56              self.sv_hist.append(("Step 7", self.new_state_variables))
57          if self.new_state_variables > 0.:
58              self.iters += 1
59              yield env.process(self.p2(env))
60          else:
61              yield env.timeout(0)
62
63
64  env = simpy.Environment()
65  print("creating environment for:\n::SA Model Development::")
66  smd = SMD(env, 10)
67  print("smd created, running sim...")
68  env.run()
69  print('end sim')
70  print('{}'.format(smd.sv_hist))
71  print(smd.iters)
```

Listing A.8 is the final version of the simulator from Experiment 2, which can run many formulations of SE methods.

Listing A.8: Generic DES Simulator

```
1  """
2  Second attempt at a general DES from a file specification
3  """
4  import simpy
5  import random
```

```
6  import operator
7  import json
8  import argparse
9
10 ACTIONS_REGISTER = {
11         "print": lambda x: print(x),
12         "add": lambda x, y: x + y,
13         "increment": lambda x: x + 1,
14         "uniformAddInt": lambda x, y, z: x + int(random.uniform(y, z)),
               # , y, z
15         "success": lambda x: x.success(),
16         "yieldtime": lambda x, en: (yield en.timeout(int(x))),
17         "nextproc": lambda f, en, passval=None: (yield en.process(f(en,
               passval=passval))),
18         "condition": None,
19         }
20
21
22 def getNode(obj, tobj, env):
23     #print("Calling node: {}".format(tobj.id))
24     if not (obj.dat is None):
25         names = ['passval','minval','maxval']
26         datalist = [obj.dat[k] for k in names if k in obj.dat.keys()]
27         if len(datalist) > 2:
28             #print("Sending a passval: {}".format([v for _,v in obj.dat.
                   items()][0]))
29             assert len(datalist) > 2
30             pval = ACTIONS_REGISTER["uniformAddInt"](*list(datalist))
31             return lambda x: ACTIONS_REGISTER['nextproc'](x, env,
                   passval=pval)
32         else:
33             #print("short data list")
34             #print("Sending a passval: {}".format(datalist[0]))
```

```
35              return lambda x: ACTIONS_REGISTER['nextproc'](x, env,
                    passval=datalist[0])
36      else:
37          return lambda x: ACTIONS_REGISTER['nextproc'](x, env, passval=
                None)
38
39  class ProcessNode:
40      def __init__(self, procid, descr, actions, nextid, dat=None):
41          self.id = procid
42          self.descr = descr
43          self.acts = actions
44          self.nextid = nextid
45          self.prev = None
46          self.next = None
47          self.dat = dat
48          self.res = []
49
50      def __call__(self, env, **kwargs):
51          if not (self.dat is None):
52              for k in self.dat.keys():
53                  if k in kwargs.keys():
54                      self.dat[k] = kwargs[k]
55          yield env.process(self.generate_tasks(env))
56
57      def act_on_it(self, env, thing):
58          for key in thing.keys():
59              if key in ACTIONS_REGISTER.keys():
60                  arg, fcn = self.parse(key, thing[key], env)
61                  assert not (arg is None)
62                  if fcn is None:
63                      if type(arg) is list:
64                          resval = ACTIONS_REGISTER[key](*arg)
65                          self.res.append(resval)
```

```
66                      else:
67                          self.res.append(ACTIONS_REGISTER[key](arg))
68                  else:
69                      yield env.process(fcn(arg))
70              else:
71                  yield env.timeout(0)
72
73      def generate_tasks(self, env, act_override=None):
74          if act_override is not None:
75              for act in act_override:
76                  yield env.process(self.act_on_it(env, act))
77          else:
78              for act in self.acts:
79                  yield env.process(self.act_on_it(env, act))
80
81      def parse(self, action_name, action_body, env):
82          if action_name == 'yieldtime':
83              action_body = int(action_body)
84          if action_body == "descr":
85              return self.descr, None
86          elif (type(action_body) is int) or (type(action_body) is float):
                #(action_name == 'yieldtime') or
87              return action_body, lambda x: ACTIONS_REGISTER['yieldtime'](
                    int(x), env)
88          elif (action_name == "nextproc") and (action_body != "nextid"):
89              assert action_body != self.id
90              found_id = False
91              thisobj = self
92              if action_body < self.id:
93                  while not found_id:
94                      thisobj = thisobj.prev
95                      thisid = thisobj.id
96                      found_id = (thisid == action_body)
```

```
97              else:
98                  while not found_id:
99                      thisobj = thisobj.next
100                     thisid = thisobj.id
101                     found_id = (thisid == action_body)
102             nfunc = getNode(self, thisobj, env)
103             return thisobj, nfunc
104         elif action_body == "nextid":
105             thisobj = self
106             nfunc = getNode(thisobj, self.next, env)
107             return self.next, nfunc
108         elif action_body == "passval,minval,maxval":
109             thevars = action_body.split(',')
110             datalist = [self.dat[k] for k in thevars if k in self.dat.
                    keys()]
111             return list(datalist), None #ACTIONS_REGISTER['uniformAddInt
                    ']
112         elif action_name == "condition":
113             test_expr = action_body["test"].copy()
114             test_tuple = test_expr.popitem()
115             body_tuple = test_tuple[1].copy().popitem()
116             control_var = self.dat[test_tuple[0]]
117             test_res = getattr(operator, body_tuple[0])(control_var,
                    body_tuple[1])
118             if test_res:
119                 true_cond_acts = action_body["iftrue"]
120                 return true_cond_acts, lambda x: (yield env.process(self
                        .generate_tasks(env, x)))
121             else:
122                 false_cond_acts = action_body["iffalse"]
123                 return false_cond_acts, lambda x: (yield env.process(
                        self.generate_tasks(env, x)))
124
```

```
125             else:
126                 print(action_body)
127                 print(type(action_body))
128                 return None, None
129
130         def __repr__(self):
131             return 'Node(%s)' % self.id
132
133         def set_prev(self, obj):
134             self.prev = obj
135
136         def set_next(self, obj):
137             self.next = obj
138
139
140     class GeneralDES:
141         def __init__(self, env, nodeData, **kwargs):
142             self.env = env
143             self.nodeData = nodeData
144             self.nodeChain = []
145             self.first = None
146             self.rootproc = None
147             self.__dict__.update(kwargs)
148
149         def make_nodes(self):
150             for node, dat in self.nodeData.items():
151                 self.nodeChain.append(ProcessNode(node, dat['descr'], dat['
                        actions'], dat['nextid'],
152                                                   dat['data'] if 'data' in
                                                          dat.keys() else None))
153             self.nodeChain.sort(key=lambda x: x.id)
154             numnodes = len(self.nodeChain)
155             assert numnodes >= 1
```

```
156             if numnodes > 1:
157                 for ndx in range(numnodes):
158                     if ndx == 0:
159                         self.nodeChain[ndx].set_next(self.nodeChain[ndx+1])
160                     elif ndx == (numnodes - 1):
161                         self.nodeChain[ndx].set_prev(self.nodeChain[ndx-1])
162                     else:
163                         self.nodeChain[ndx].set_next(self.nodeChain[ndx+1])
164                         self.nodeChain[ndx].set_prev(self.nodeChain[ndx-1])
165             self.first = self.nodeChain[0]
166             self.rootproc = self.env.process(self.first(self.env))
167 
168         def generic_function(self, identity, **kwargs):
169             myid = identity
170             if 'actions' in self.__dict__.keys():
171                 pass
172 
173 
174 parser = argparse.ArgumentParser()
175 parser.add_argument("file")
176 args = parser.parse_args()
177 nodeData = None
178 with open(args.file, "r") as tf:
179     nodeData = json.load(tf)
180 if nodeData is not None:
181     env = simpy.Environment()
182     gendes = GeneralDES(env, nodeData)
183     gendes.make_nodes()
184     print(gendes.nodeChain)
185     env.run()
186     print("RAN")
187     print(gendes.nodeChain[-1].res)
```

The input file for SRD for the simulator is Listing A.9, and the input file for SAMD is Listing A.10.

Listing A.9: SRD Input File

```
{
  "p1": {
    "descr": "1.  Defining external interfaces for mission elements
       ...",
    "actions": [
      {
        "print": "descr"
      },
      {
        "uniformAddInt": "passval,minval,maxval"
      },
      {
        "nextproc": "nextid"
      }
    ],
    "nextid": "p2",
    "data": {
      "passval": 0,
      "minval": 5,
      "maxval": 10
    }
  },
  "p2": {
    "descr": "2.  Specifying spacecraft behavior supporting required
       functions and states",
    "actions": [
      {
        "print": "descr"
      },
```

```
      {
        "uniformAddInt": "passval,minval,maxval"
      },
      {
        "nextproc": "nextid"
      }
    ],
    "nextid": "p3",
    "data": {
      "passval": 0,
      "minval": 5,
      "maxval": 10
    }
  },
  "p3": {
    "descr": "3.  Identifying potential failure modes",
    "actions": [
      {
        "print": "descr"
      },
      {
        "uniformAddInt": "passval,minval,maxval"
      },
      {
        "nextproc": "nextid"
      }
    ],
    "nextid": "p4",
    "data": {
      "passval": 0,
      "minval": 5,
      "maxval": 10
    }
```

```
},
"p4": {
  "descr": "4.  Specifying spacecraft performance, physical, and
     quality characteristics",
  "actions": [
    {
      "print": "descr"
    },
    {
      "uniformAddInt": "passval,minval,maxval"
    },
    {
      "nextproc": "nextid"
    }
  ],
  "nextid": "p5",
  "data": {
    "passval": 0,
    "minval": 5,
    "maxval": 10
  }
},
"p5": {
  "descr": "5.  Capture system requirements in terms of a black
     box specification to refine text-based requirements",
  "actions": [
    {
      "print": "descr"
    },
    {
      "uniformAddInt": "passval,minval,maxval"
    },
    {
```

```
        "yieldtime": 0
      }
    ],
    "nextid": null,
    "data": {
      "passval": 0,
      "minval": 5,
      "maxval": 10
    }
  }
}
```

Listing A.10: SAMD Input File

```
{
  "p1": {
    "descr": "1.  Identify Needs",
    "actions": [
      {
        "print": "descr"
      },
      {
        "nextproc": "nextid"
      }
    ],
    "nextid": "p2",
    "data": {
      "passval": 10,
      "minval": 0,
      "maxval": 0
    }
  },
  "p2": {
```

```
    "descr": "2.  Identify State Variables",
    "actions": [
      {
        "print": "descr"
      },
      {
        "uniformAddInt": "passval,minval,maxval"
      },
      {
        "nextproc": "nextid"
      }
    ],
    "nextid": "p3",
    "data": {
      "passval": 0,
      "minval": 0,
      "maxval": 2
    }
  },
  "p3": {
    "descr": "3.  Define State Models for State Variables",
    "actions": [
      {
        "print": "descr"
      },
      {
        "uniformAddInt": "passval,minval,maxval"
      },
      {
        "nextproc": "nextid"
      }
    ],
    "nextid": "p4",
```

```
    "data": {
      "passval": 0,
      "minval": -10,
      "maxval": -2
    }
  },
  "p4": {
    "descr": "4.  Identify Measures for State Estimation",
    "actions": [
      {
        "print": "descr"
      },
      {
        "nextproc": "nextid"
      }
    ],
    "nextid": "p5",
    "data": {
      "passval": 0,
      "minval": 0,
      "maxval": 0
    }
  },
  "p5": {
    "descr": "5.  Define Measurement Models",
    "actions": [
      {
        "print": "descr"
      },
      {
        "uniformAddInt": "passval,minval,maxval"
      },
      {
```

```
        "nextproc": "nextid"
      }
    ],
    "nextid": "p6",
    "data": {
      "passval": 0,
      "minval": 0,
      "maxval": 4
    }
  },
  "p6": {
    "descr": "6.  Identify commands to control state variables",
    "actions": [
      {
        "print": "descr"
      },
      {
        "nextproc": "nextid"
      }
    ],
    "nextid": "p7",
    "data": {
      "passval": 0,
      "minval": 0,
      "maxval": 0
    }
  },
  "p7": {
    "descr": "7.  Define command models",
    "actions": [
      {
        "print": "descr"
      },
```

```
119       {
120         "condition": {
121           "test": {
122             "passval": {
123               "gt": 0
124             }
125           },
126           "iftrue": [
127             {
128               "uniformAddInt": "passval,minval,maxval"
129             },
130             {
131               "nextproc": "p2"
132             }
133           ],
134           "iffalse": [
135             {
136               "yieldtime": 0
137             }
138           ]
139         }
140       }
141     ],
142     "nextid": null,
143     "data": {
144       "passval": 0,
145       "minval": 0,
146       "maxval": 4
147     }
148   }
149 }
```

Listing A.11 provided the capability to automate running the simulator for Monte

Carlo simulation in Experiment 2.

Listing A.11: Autorun Script for DES Monte Carlo

```
import numpy as np
import subprocess

PRIMARY_KEYS = ['pythonpath', 'scriptpath', 'jsonfile']

class CommandWrapper:
    def __init__(self, **kwargs):
        self.args = {}
        for k, v in kwargs.items():
            if k in PRIMARY_KEYS:
                self.args[k] = v

    def run_proc(self, n):
        """
        Run Process n times, return generator of CompletedProcess
        """
        flat_commands = []
        for k in PRIMARY_KEYS:
            flat_commands.append(self.args[k])
        num = 0
        while num < n:
            yield subprocess.run(flat_commands, capture_output=True)
            num += 1

test_runs = 1000
pythonpath = "python.exe"
scriptpath = "generalDES-2.py"
jsonfile = "oosem.json"
testWrap = CommandWrapper(pythonpath=pythonpath, scriptpath=scriptpath,
    jsonfile=jsonfile)
```

```
res_oosem = [r for r in testWrap.run_proc(test_runs)]
np.savez_compressed("out1.npy", result=res_oosem, allow_pickle=True)
print(len(res_oosem))

jsonfile = "sta.json"
testWrap = CommandWrapper(pythonpath=pythonpath, scriptpath=scriptpath,
    jsonfile=jsonfile)
res_sta = [r for r in testWrap.run_proc(test_runs)]
np.savez_compressed("out2.npy", result=res_sta, allow_pickle=True)
print(len(res_sta))
```

Finally, the data from the automatic runs were plotted using Listing A.12.

Listing A.12: Plotting DES Monte Carlo Data

```
import ast
import numpy as np
import matplotlib.pyplot as plt


def indexOfRAN(alist):
    strTarget = u'RAN'
    ndx = 0
    breakingNum = 10000
    isNotFound = True
    while isNotFound:
        isNotFound = not (strTarget in alist[ndx])
        if isNotFound:
            ndx += 1
        if ndx > breakingNum:
            print('Too Many Iterations')
            return None
    return ndx

```

```
20 thedata1 = np.load("out1.npy.npz", allow_pickle=True)
21 thedata2 = np.load("out2.npy.npz", allow_pickle=True)
22
23 # Analyze the data from OOSEM-Lite (Architecting Spacecraft)
24 # Get statistics on the amount of time
25 caselist = thedata1["result"]
26 timevals = []
27 for acase in caselist:
28     case_str_data = acase.stdout.decode('ascii').split('\r\n')
29     targetNdx = indexOfRAN(case_str_data)
30     resultTuple = ast.literal_eval(case_str_data[targetNdx+1])
31     assert isinstance(resultTuple, list)
32     assert len(resultTuple) == 2
33     timevals.append(resultTuple[1])
34
35 numSamples = len(timevals)
36 meanTime = float(sum(timevals))/float(numSamples)
37 stdDev = np.std(timevals)
38 print("Data about OOSEM-Lite, SRD")
39 print("Number of samples: {}".format(numSamples))
40 print("Mean Duration: {}".format(meanTime))
41 print("Deviation: {}".format(stdDev))
42 fig, ax = plt.subplots(1, 1)
43 ax.set_title('Durations for SRD')
44 red_square = dict(markerfacecolor='r', marker='s')
45 ax.boxplot(np.asarray(timevals), vert=False, flierprops=red_square)
46 plt.show()
47
48 # Analyze the data from State Analysis Model Development
49 # Get statistics on the amount of time
50 caselist = thedata2["result"]
51 sv_vals = []
52 numtasks = []
```

```
53 for acase in caselist:
54     case_str_data = acase.stdout.decode('ascii').split('\r\n')
55     targetNdx = indexOfRAN(case_str_data)
56     completedSteps = case_str_data[0:(targetNdx-1)]
57     numtasks.append(sum(u'2. ' in s for s in completedSteps))
58     #print(case_str_data[targetNdx+1])
59     resultTuple = ast.literal_eval(case_str_data[targetNdx+1])
60     assert isinstance(resultTuple, list)
61     assert len(resultTuple) > 0
62     sv_vals.append([k for k in resultTuple if k is not None])
63
64 numSamples = len(numtasks)
65 meanSViter = float(sum(numtasks))/float(numSamples)
66 stdDev = np.std(numtasks)
67 print("Data about SA-MD")
68 print("Number of samples: {}".format(numSamples))
69 print("Mean Iterations: {}".format(meanSViter))
70 print("Deviation: {}".format(stdDev))
71 fig, ax = plt.subplots(1, 1)
72 ax.set_title('Iterations for SA-MD')
73 ax.boxplot(np.asarray(numtasks), vert=False, flierprops=red_square)
74 plt.show()
75
76 fig, ax = plt.subplots(1, 1)
77 ax.set_title('State Variables for SA-MD')
78 ax.hist(np.concatenate(sv_vals), density=True, alpha=0.5)
79 plt.show()
80
81 fig, ax = plt.subplots(1, 1)
82 ax.set_title('State Variables Histories for SA-MD')
83 for data in sv_vals:
84     data[len(data):] = [0]
85     ax.plot(range(0, len(data)), data, 'k.-', linewidth=0.5, markersize
```

```
        =1)
ax.set_xlabel('Update Number')
ax.set_ylabel('State Variables')
plt.show()
```

A.3 PMF Formulation

Experiment 4 utilized a formulation of methods by probability mass function convolution. Listing A.13 was the means for that analysis.

Listing A.13: PMF Convolution Study for Experiment 4, SAMD

```
import matplotlib.pyplot as plt
import numpy.fft

def convolve_many(arrays):
    """
    Convolve a list of 1d float arrays together, using FFTs.
    The arrays need not have the same length, but each array should
    have length at least 1.
    https://stackoverflow.com/questions/28901221/faster-convolution-of-
        probability-density-functions-in-python/29236193#29236193
    """
    result_length = 1 + sum((len(array) - 1) for array in arrays)

    # Copy each array into a 2d array of the appropriate shape.
    rows = numpy.zeros((len(arrays), result_length))
    for i, array in enumerate(arrays):
        rows[i, :len(array)] = array

    # Transform, take the product, and do the inverse transform
    # to get the convolution.
    fft_of_rows = numpy.fft.fft(rows)
```

```
21      fft_of_convolution = fft_of_rows.prod(axis=0)
22      convolution = numpy.fft.ifft(fft_of_convolution)
23
24      # Assuming real inputs, the imaginary part of the output can
25      # be ignored.
26      return convolution.real
27
28
29  class ProbDist:
30      def __init__(self):
31          self.vals = []
32          self.dist = []
33
34  class URV(ProbDist):
35      # Generate Uniform Random Variable Probability Mass (dist)
36      def __init__(self, a, b):
37          super(URV, self).__init__()
38          assert isinstance(a,int)
39          assert isinstance(b,int)
40          assert a<b
41          self.a = a
42          self.b = b
43          self.makeVals()
44          self.applyDist()
45
46      def makeVals(self):
47          v = self.a
48          while v <= self.b:
49              self.vals.append(v)
50              v += 1
51
52      def applyDist(self):
53          prob = 1./(self.b - self.a + 1.)
```

```
54          for v in self.vals:
55              self.dist.append((v,prob))
56
57      def sample(self,x):
58          assert isinstance(x,int)
59          if x<self.a or x>self.b:
60              return 0.
61          else:
62              assert len(self.dist)>0
63              for val in self.dist:
64                  if x == val[0]:
65                      return val[1]
66
67
68  class Conv(ProbDist):
69      # convolution
70      def __init__(self, *argv):
71          super(Conv, self).__init__()
72          self.dlist = [arg for arg in argv]
73          self.convolve()
74
75      def convolve(self):
76          dists = [d.dist for d in self.dlist]
77          nums = [[p[1] for p in d] for d in dists]
78          min_integer = sum([min([p[0] for p in d]) for d in dists])
79          max_integer = sum([max([p[0] for p in d]) for d in dists])
80          xvals = [x for x in range(min_integer, max_integer+1)]
81          weights = convolve_many(nums)
82
83          for i, val in enumerate(xvals):
84              self.vals.append(val)
85              self.dist.append((val,weights[i]))
86
```

```
87      def add_scalar(self, q):
88          output = ProbDist()
89          for x in range(len(self.vals)):
90              output.vals.append(self.vals[x] + q)
91          for x in range(len(self.dist)):
92              output.dist.append((self.dist[x][0] + q, self.dist[x][1]))
93          return output
94
95
96  def calcProbs(q, tol, splitTuple, *argv):
97      # from zero to qmax
98      procDists = []
99      assert len(splitTuple) == 2
100     for num in splitTuple:
101         convDists = []
102         for x in range(num):
103             convDists.append(argv[x])
104         thisD = Conv(*convDists)
105         procDists.append(thisD)
106     k = 0
107     kvals = [0]
108     pg0list = []
109     while True:
110         if k > 100:
111             return kvals, pg0list
112         if k == 0:
113             prevDist = procDists[0].add_scalar(q)
114             pg0 = sum([p[1] for p in prevDist.dist if p[0] > 0])
115             pg0list.append(pg0)
116         elif k > 0:
117             extraDists = [procDists[1]]*k
118             thisDist = Conv(prevDist, *extraDists)
119             prevDist = thisDist
```

```
120                 pg0 = sum([p[1] for p in prevDist.dist if p[0] > 0])
121                 if abs(pg0 - pg0list[-1]) < tol:
122                     pg0list.append(pg0)
123                     return kvals, pg0list
124                 pg0list.append(pg0)
125             k += 1
126             kvals.append(k)
127
128
129 # SAMD Parameterization
130 initial_State_Variables = 10
131 tol = 1.e-5 # tolerance to stop computing the cdf
132 caseNodes = [
133     [URV(0,2), URV(-10,-2), URV(0,4), URV(0,4)],
134     [URV(0,2), URV(-5,-2), URV(0,4), URV(0,4)],
135     [URV(0,2), URV(-10,-5), URV(0,4), URV(0,4)],
136     [URV(0,2), URV(-10,-2), URV(0,4), URV(0,4)],
137     [URV(0,2), URV(-10,-2), URV(0,4), URV(0,4)],
138     [URV(0,2), URV(-10,-2), URV(0,4), URV(0,4)],
139     [URV(0,2), URV(-10,-2), URV(0,4), URV(0,4)]
140 ]
141 caseSelections = [
142     (3,4),
143     (3,4),
144     (3,4),
145     (2,4),
146     (1,4),
147     (1,3),
148     (1,2)
149 ]
150 caseNames = [
151     'Default Case',
152     'Case A',
```

```
153         'Case␣B',
154         'Case␣C',
155         'Case␣D',
156         'Case␣E',
157         'Case␣F'
158     ]
159     markers = [
160         's',
161         'o',
162         'h',
163         '^',
164         'x',
165         '*',
166         '+'
167     ]
168     xyvals = []
169     for i, acase in enumerate(caseNodes):
170         thesekvals, thesepvals = calcProbs(initial_State_Variables, tol,
                caseSelections[i], *acase)
171         xyvals.extend([thesekvals,thesepvals,'{}:'.format(markers[i])])
172
173     fig, ax = plt.subplots()
174     ax.set_title('Iteration␣vs␣Probability␣Number␣of␣State␣Variables␣is␣>␣0'
            )
175     plt.plot(*xyvals)
176     plt.legend(caseNames)
177     plt.ylabel('Probability')
178     plt.xlabel('Iteration␣Number')
179     plt.show()
```

A.4 System Analysis Codes

Experiment 5 required some system analysis codes. In particular, Listing A.14 contains some antenna and link analysis, with the ability run DoE and noise variable Monte Carlo. Listing A.15 provides the genericized DoE generation capability. These codes can be run from command line or from the system model.

Listing A.14: Antenna and Link Analysis

```
1  import sys
2  import argparse
3  import json
4  import csv
5  import numpy as np
6  from os import path
7  from numpy.random import default_rng
8  from scipy.stats import gamma
9
10 # some inputs are DoE variables, some are random, rest are constants
11
12
13 class RandomVariate:
14
15     def __init__(self, varname, distname, params):
16         self.name = varname
17         self.rng = default_rng()
18         self.distMap = {
19             'uniform': lambda x, L, H: self.rng.uniform(L,H,x),
20             'normal': lambda x, m, s: self.rng.normal(m,s,x),
21             'triangular': lambda x, l, m, r: self.rng.triangular(l,m,r,x
                   )
22         }
23         assert distname in self.distMap.keys()
```

```
24          if distname=='uniform' or distname=='normal':
25              assert len(params) == 2
26          else:
27              assert len(params) == 3
28          self.dist = distname
29          self.distparams = params
30
31      def sample(self, x=1):
32          return self.distMap[self.dist](x, *self.distparams).tolist()
33
34
35  class LinkCalculation:
36
37      PI = np.pi
38      C = 3.*10.**8. # speed of light m/s?
39      # input: frequency (GHz)
40      # input: transmitter power (W)
41      # calc: transmitter power (dBW)
42
43      def __init__(self, **kwargs):
44          self.variables = []
45          self.definitions = {}
46          for k, v in kwargs.items():
47              self.variables.append(k)
48              self.definitions[k] = v
49          assert 'f' in self.variables # input: frequency (GHz)
50          assert 'P' in self.variables # input: transmitter power (W)
51          assert 'Ll' in self.variables # input: line loss (dB)
52          """
53          SAMD 3rd edition has beamwidth as an input, thereby calculating
                antenna diameter and peak gain.
54
55          However, Gross establishes a "transmitter efficiency" which is
```

```
        in degrees
56
57      The efficiency variables of Gross appear to play the role that
            beam widths and offsets play in SMAD.
58
59      However, since antenna gain is calculated from the antenna
            design, and not the halfpower beamwidth, the
60      antenna gain is an input here instead of an output.
61      """
62      assert 'Gant' in self.variables # antenna gain (dBi)
63      assert 't_eff' in self.variables # gross transmitter efficiency
            (deg) as beamwidth (deg)
64      assert 'p_off' in self.variables # pointing offset (deg)
65      assert 'S' in self.variables # path length (km)
66      assert 'La' in self.variables # propagation and polarization
            loss (dB) from atmospheric properties
67      assert 'Dr' in self.variables # receive antenna diameter (m)
68      assert 'eta_gr' in self.variables # receive antenna efficiency
            (0,1)
69      assert 'err_rec_pt' in self.variables # receive antenna pointing
             error (deg)
70      assert 'Ts' in self.variables # system noise temperature (K)
71      assert 'r' in self.variables # data rate (bps)
72      #assert 'ber' in self.variables # bit error rate
73      assert 'impl_loss' in self.variables # implementation loss (dB)
74      assert 'req_ebno' in self.variables # requirement from table
            based on BER (dB)
75
76
77      self.transmitPowerDB = 0. # calc: transmitter power (dBW)
78      self.pointingLoss = 0. # calc: pointing loss (dB)
79      self.netTransAntGain = 0. # calc: net transmit antenna gain (dBi
            )
```

```
80          self.eirp = 0. # calc: equivalent isotropic radiated power (dBW)
81          self.spaceLoss = 0. # calc: space loss over path length (dB)
82          self.recAntGainNet = 0. # calc: receive antenna net peak gain (
                dBi)
83          self.recBeamWidth = 0. # calc: receive antenna half power beam
                width (deg)
84          self.recPointLoss = 0. # calc: receive antenna pointing loss (dB
                )
85          self.recAntGain = 0. # calc: receive antenna gain (dBi)
86          self.ebno = 0. # calc: energy per bit to noise density (dB)
87          self.cno = 0. # calc: carrier to noise density ratio (dB-Hz)
88          self.margin = 0. # calc: link margin (dB)
89
90          self.outputs = {
91              'transmitPowerDB': self.transmitPowerDB,
92              'pointLoss': self.pointingLoss,
93              'G_t_net': self.netTransAntGain,
94              'eirp': self.eirp,
95              'Ls': self.spaceLoss,
96              'Grp': self.recAntGainNet,
97              'th_g': self.recBeamWidth,
98              'Lpr': self.recPointLoss,
99              'Gr': self.recAntGain,
100             'EBNO': self.ebno,
101             'CNO': self.cno,
102             'margin': self.margin
103         }
104
105     def run_calculation(self):
106         self.outputs['transmitPowerDB'] = self.calcTPDB()
107         self.outputs['pointLoss'] = self.calcPointingLoss()
108         self.outputs['G_t_net'] = self.calcNetTGain()
109         self.outputs['eirp'] = self.calcEIRP()
```

```
110         self.outputs['Ls'] = self.calcSpaceLoss()
111         self.outputs['Grp'] = self.calcNetRGain()
112         self.outputs['th_g'] = self.calcRecBeamWidth()
113         self.outputs['Lpr'] = self.calcRecPtLoss()
114         self.outputs['Gr'] = self.calcRecAntGain()
115         self.outputs['EBNO'] = self.calcEBNO()
116         self.outputs['CNO'] = self.calcCNO()
117         self.outputs['margin'] = self.calcMargin()
118
119     def calcTPDB(self):
120         transmitterPower = self.definitions['P']
121         self.transmitPowerDB = 10.*np.log10(transmitterPower)
122         return self.transmitPowerDB
123
124     def calcPointingLoss(self):
125         offset = self.definitions['p_off']
126         width = self.definitions['t_eff']
127         self.pointingLoss = -12. * ((offset/width)**2.)
128         return self.pointingLoss
129
130     def calcNetTGain(self):
131         antenna_gain = self.definitions['Gant']
132         pointing_loss = self.pointingLoss
133         self.netTransAntGain = antenna_gain + pointing_loss
134         return self.netTransAntGain
135
136     def calcEIRP(self):
137         transmit_power_db = self.transmitPowerDB
138         line_loss = self.definitions['Ll']
139         net_gain = self.netTransAntGain
140         self.eirp = transmit_power_db + line_loss + net_gain
141         return self.eirp
142
```

```
143     def calcSpaceLoss(self):
144         freq = self.definitions['f']*10.**9.
145         space = self.definitions['S']*10.**3.
146         self.spaceLoss = 20.*np.log10(self.C) - 20.*np.log10(4.*self.PI)
                - 20.*np.log10(space) - 20.*np.log10(freq)
147         return self.spaceLoss
148
149     def calcNetRGain(self):
150         ground_diameter = self.definitions['Dr']
151         ground_eff = self.definitions['eta_gr']
152         freq = self.definitions['f']*10.**9.
153         self.recAntGainNet = 20.*np.log10(self.PI) + 20.*np.log10(
              ground_diameter) + 20.*np.log10(freq) \
154                                 + 20.*np.log10(ground_eff) - 20.*np.log10(
                                      self.C)
155         return self.recAntGainNet
156
157     def calcRecBeamWidth(self):
158         ground_diameter = self.definitions['Dr']
159         freq = self.definitions['f'] # GHz
160         self.recBeamWidth = 21./(freq*ground_diameter)
161         return self.recBeamWidth
162
163     def calcRecPtLoss(self):
164         error = self.definitions['err_rec_pt']
165         beamwidth = self.recBeamWidth
166         self.recPointLoss = -12.*((error/beamwidth)**2.)
167         return self.recPointLoss
168
169     def calcRecAntGain(self):
170         netGain = self.recAntGainNet
171         ptloss = self.recPointLoss
172         self.recAntGain = netGain + ptloss
```

```
173         return self.recAntGain
174
175     def calcEBNO(self):
176         temp_s = self.definitions['Ts']
177         data_rate = self.definitions['r']
178         self.ebno = self.eirp + self.recPointLoss + self.spaceLoss +
                self.recAntGain + 228.6 - 10.*np.log10(temp_s) \
179                     - 10.*np.log10(data_rate)
180         return self.ebno
181
182     def calcCNO(self):
183         data_rate = self.definitions['r']
184         self.cno = self.ebno + 10.*np.log10(data_rate)
185         return self.cno
186
187     def calcMargin(self):
188         ebno = self.ebno
189         reqebno = self.definitions['req_ebno']
190         iloss = self.definitions['impl_loss']
191         self.margin = ebno - reqebno + iloss
192         return self.margin
193
194     def refresh_variables(self, **kwargs):
195         for k, v in kwargs.items():
196             if k in self.definitions.keys():
197                 self.definitions[k]=v
198         self.run_calculation()
199
200
201 class AntennaCalculation:
202
203     PI = np.pi
204
```

```
205     def __init__(self, **kwargs):
206         self.variables = []
207         self.definitions = {}
208         for k,v in kwargs.items():
209             self.variables.append(k)
210             self.definitions[k] = v
211         self.gain = 0.
212         self.mass = 0.
213         self.outputs = {'gain':self.gain, 'mass':self.mass}
214 
215     def calculate_gain(self):
216         return self.gain
217 
218     def calculate_mass(self):
219         return self.mass
220 
221     def run_calculation(self):
222         self.outputs['gain'] = self.calculate_gain()
223         self.outputs['mass'] = self.calculate_mass()
224 
225     def refresh_variables(self, **kwargs):
226         for k, v in kwargs.items():
227             if k in self.definitions.keys():
228                 self.definitions[k]=v
229         self.run_calculation()
230 
231 
232 class ParabolicCalculation(AntennaCalculation):
233     def __init__(self, **kwargs):
234         super().__init__(**kwargs)
235         assert 'd' in self.variables
236         assert 'lambd' in self.variables
237         assert 'h' in self.variables
```

```
238         assert 'rho' in self.variables
239         self.run_calculation()
240
241     def calculate_gain(self):
242         d = self.definitions['d']
243         lam = self.definitions['lambd']
244         self.gain = np.log10((self.PI**2.)*(d**2.)*(1./(lam**2.)))
245         return self.gain
246
247     def calculate_mass(self):
248         d  = self.definitions['d']
249         h  = self.definitions['h']
250         rh = self.definitions['rho']
251         self.mass = (d**2.)*(self.PI/4.)*h*rh
252         return self.mass
253
254
255 class HornCalculation(AntennaCalculation):
256
257     def __init__(self, **kwargs):
258         super().__init__(self, **kwargs)
259         assert 'shape' in self.variables
260         assert (self.definitions['shape']=='pyramidal') or (self.
                definitions['shape']=='conical')
261         if self.definitions['shape'] == 'pyramidal':
262             assert 'Le' in self.variables
263             assert 'Lh' in self.variables
264             self.isPyramid = True
265         else:
266             assert 'L' in self.variables
267             self.isPyramid = False
268         assert 'lambd' in self.variables
269         assert 'ea' in self.variables
```

```
270             assert 'h' in self.variables
271             self.A = 0.
272             self.run_calculation()
273
274         def calculate_gain(self):
275             lam = self.definitions['lambd']
276             ea = self.definitions['ea']
277             if self.isPyramid:
278                 Le = self.definitions['Le']
279                 Lh = self.definitions['Lh']
280                 alpha_e = np.sqrt(2.*lam*Le)
281                 alpha_h = np.sqrt(2.*lam*Lh)
282                 self.A = alpha_e * alpha_h
283                 self.gain = (4.*self.PI*self.A)*ea/(lam**2.)
284             else:
285                 L = self.definitions['L']
286                 d = np.sqrt(3.*lam*L)
287                 self.A = self.PI * ((d/2.)**2.)
288                 self.gain = (((self.PI*d)/lam)**2.)*ea
289             self.gain = np.log10(self.gain)
290             return self.gain
291
292         def calculate_mass(self):
293             # https://ocw.mit.edu/courses/aeronautics-and-astronautics
                    /16-851-satellite-engineering-fall-2003/assignments/
                    ps4_cs_solution.pdf
294             h = self.definitions['h']
295             effective_d = np.sqrt(self.A/self.PI)*2.
296             self.mass = 16*h*effective_d
297             return self.mass
298
299 # Need: DoE Variables
300 # Need: DoE Variable Settings (usual DoE matrix)
```

```
301 # --> refactor DoE tool to output csv to get headers + cases
302
303 def horn_initializer(*args):
304     assert len(args)==5 or len(args)==6
305     shp = args[0]
306     lambd = args[1]
307     ea = args[2]
308     h = args[3]
309     if shp=='conical':
310         L = args[4]
311         return HornCalculation(shape=shp, L=L, lambd=lambd, ea=ea, h=h)
312     elif shp=='pyramidal':
313         Le = args[4]
314         Lh = args[5]
315         return HornCalculation(shape=shp, Le=Le, Lh=Lh, lambd=lambd, ea=
                ea, h=h)
316     else:
317         return None
318
319
320 ANTENNA_MODELS = {
321         "Parabolic": lambda d, lam, h, rho: ParabolicCalculation(d=d,
                lambd=lam, h=h, rho=rho),
322         "Horn": lambda args: horn_initializer(*args)
323         }
324
325
326 def run_cases_with_mixed_vars(model, doe, consts, rvs, outvar, mcsize):
327     output_data = []
328     datastore = []
329     marginvals = []
330     for case_num, case in enumerate(doe):
331         outputsamples = []
```

```
332             mc_count = 1
333             while mc_count < mcsize:
334                 rvars = {x.name: x.sample(1)[0] for x in rvs}
335                 variables = {**case, **consts, **rvars}
336                 model.refresh_variables(**variables)
337                 outputsamples.append(model.outputs[outvar])
338                 datastore.append({**{'case': case_num, 'rep': mc_count}, **
                        variables, **model.outputs})
339                 mc_count += 1
340             marginvals.extend(outputsamples)
341             output_data.append(list(gamma.fit(outputsamples)))
342         print(max(marginvals))
343         print(min(marginvals))
344         print(np.mean(marginvals))
345         return list(output_data), list(datastore)
346
347
348 def read_doe_csv(doe_file_path):
349     cases = []
350     with open(doe_file_path, 'r') as csvfile:
351         reader = csv.DictReader(csvfile)
352         for row in reader:
353             cases.append({k:float(v) for k, v in row.items()})
354     return list(cases)
355
356
357 def create_rvs_from_json(rv_json):
358     fd = None
359     rv_list = []
360     #with open(rv_json_fp, 'r') as tf:
361     #    fd = json.load(tf)
362     fd = json.loads(rvs_json)
363     if fd is not None:
```

```
364             for k, v in fd.items():
365                 rv_list.append(RandomVariate(k, v['distribution'], v['params
                        ']))
366         return list(rv_list)
367
368
369     def parse_doe_string(doe):
370         cases = []
371         print(doe)
372         doe_rows = doe.split('\"')[1].split('\\n')
373         headers = doe_rows.pop(0).split(',')
374         for data_row in doe_rows:
375             thiscase = data_row.split(',')
376             c = {x:float(y) for x,y in zip(headers, thiscase)}
377             cases.append(c)
378         return cases
379
380
381     def run_cases(ant_model, doe):
382         # ant_model: str tuple (string specifying ANTENNA_MODEL, '' or '
                conical' or 'pyramidal')
383         # doe: csv format data?
384         output = [['gain','mass']]
385         assert ant_model[0] in ANTENNA_MODELS.keys()
386         doe_cases = parse_doe_string(doe)
387         if ant_model[0]=='Parabolic':
388             parabolic_model = ANTENNA_MODELS[ant_model[0]](doe_cases[0]["d"
                    ], doe_cases[0]["lam"], doe_cases[0]["h"], doe_cases[0]["rho
                    "])
389             for case in doe_cases:
390                 parabolic_model.refresh_variables(**case)
391                 output.append([parabolic_model.outputs['gain'],
                        parabolic_model.outputs['mass']])
```

```
392     elif ant_model[0]=='Horn' and ant_model[1]!='':
393         if ant_model[1]=='conical':
394             arg_list = [ant_model[1], doe_cases[0]["lam"], doe_cases[0][
                    "ea"], doe_cases[0]["h"], doe_cases[0]["L"]]
395         else:
396             arg_list = [ant_model[1], doe_cases[0]["lam"], doe_cases[0][
                    "ea"], doe_cases[0]["h"], doe_cases[0]["Le"], doe_cases
                    [0]["Lh"]]
397         horn_model = ANTENNA_MODELS[ant_model[0]](arg_list)
398         for case in doe_cases:
399             horn_model.refresh_variables(**case)
400             output.append([horn_model.outputs['gain'], horn_model.
                    outputs['mass']])
401     else:
402         print("Unsupported Antenna")
403     return output
404
405
406 def format_output_text(data):
407     outlist = []
408     output = ''
409     for row in data:
410         row_str = ''
411         row_list = []
412         for ndx in range(len(row)):
413             itm = row[ndx]
414             row_list.append(itm)
415             if ndx<(len(row)-1):
416                 row_str = row_str + '{}'.format(itm) + ','
417             else:
418                 row_str = row_str + '{}'.format(itm) + '\n'
419         output += row_str
420         outlist.append(list(row_list))
```

```
421     sys.stdout.write(output)
422     return list(outlist)
423
424
425 def main(arg1, arg2):
426     model_list = json.loads(arg1)
427     ant_model = tuple(model_list)
428     outlist = format_output_text(run_cases(ant_model, arg2))
429     with open('res_ant.csv', 'w') as cf:
430         cwriter = csv.writer(cf)
431         for row in outlist:
432             cwriter.writerow(row)
433
434
435 if __name__ == '__main__':
436     doe_fp = sys.argv[1]
437     consts_json = sys.argv[2]
438     rvs_json = sys.argv[3]
439     outvar = sys.argv[4]
440     num_mc = int(sys.argv[5])
441
442     doe_cases = read_doe_csv(doe_fp)
443     rvs_def = create_rvs_from_json(rvs_json)
444     consts_dict = json.loads(consts_json)
445
446     initialvars = {**doe_cases[0], **{r.name: 0. for r in rvs_def}, **
            consts_dict}
447     link_model = LinkCalculation(**initialvars)
448
449     dist_list, data = run_cases_with_mixed_vars(link_model, doe_cases,
            consts_dict, rvs_def, outvar, num_mc)
450     print(dist_list[-1])
451     print(len(dist_list))
```

```
452     doe_path_chain = doe_fp.split('\\')
453     doe_path_chain.pop(-1)
454     with open(path.join('\\'.join(doe_path_chain), 'md_data.csv'), 'w',
            newline='') as tf:
455         fields = [k for k in data[0].keys()]
456         writer = csv.DictWriter(tf, fieldnames=fields)
457         writer.writeheader()
458         writer.writerows(data)
459
460     with open(path.join('\\'.join(doe_path_chain), '{}_dists.csv'.format
            (outvar)), 'w', newline='') as tf:
461         headers = ['shape', 'loc', 'scale']
462         writer = csv.writer(tf)
463         writer.writerow(headers)
464         writer.writerows(dist_list)
```

Listing A.15: DoE Generation (Generic)

```
1  #!/usr/local/bin/python3
2  import sys
3  import json
4  import csv
5  import numpy as np
6  from os import path
7  from pyDOE2 import *
8
9
10 DOE_MAPPER = {
11     'bb': lambda x, y: bbdesign(x, center=y),
12     'cc': lambda x, y=(4, 4), a='o', b='ccc': ccdesign(x, center=y,
           alpha=a, face=b),
13     'gff': lambda x=list(): fullfact(x),
14     '2ff': lambda n=0: ff2n(n),
```

```
15      'frc': lambda g='': fracfact(g),
16      'pb': lambda x=0: pbdesign(x)
17  }
18
19
20  class DesignFactor:
21
22      def __init__(self, **kwargs):
23          self.__dict__.update(kwargs)
24          assert "imin" in self.__dict__.keys()
25          assert "imax" in self.__dict__.keys()
26          assert "name" in self.__dict__.keys()
27
28      def __repr__(self):
29          return "var::{}(min={},max={})".format(self.name, self.imin,
                self.imax)
30
31      def normalize(self, x):
32          diff = self.imax - self.imin
33          return (2. / diff) * (x - self.imin) - 1.
34
35      def denormalize(self, xn):
36          diff = self.imax - self.imin
37          return (diff / 2.) * (xn + 1.) + self.imin
38
39
40  class Experiment:
41
42      def __init__(self, facdefs, etype):
43          self.facdefs = facdefs
44          self.etype = etype
45          self.facs = []
46          self.doe = None
```

```
47
48      def build_factors(self):
49          for fac in self.facdefs:
50              self.facs.append(
51                      DesignFactor(name=fac["name"], imin=fac["min"], imax
                            =fac["max"])
52                      )
53
54      def build_doe(self, doe_spec):
55          doe_key = doe_spec["name"]
56          if doe_key=='bb':
57              num_fac = len(self.facs)
58              assert num_fac > 0
59              assert 'centers' in doe_spec.keys()
60              center_points = doe_spec["centers"]
61              self.doe = DOE_MAPPER[doe_key](num_fac, center_points)
62          elif (doe_key=='pb') or (doe_key=='2ff'):
63              num_fac = len(self.facs)
64              assert num_fac > 0
65              self.doe = DOE_MAPPER[doe_key](num_fac)
66          elif doe_key=='gff':
67              assert 'levels' in doe_spec.keys()
68              levels = doe_spec["levels"]
69              assert isinstance(levels, list) and len(levels)>0
70              self.doe = DOE_MAPPER[doe_key](levels)
71          elif doe_key=='frc':
72              assert 'symbols' in doe_spec.keys()
73              gen = doe_spec["symbols"]
74              self.doe = DOE_MAPPER[doe_key](gen)
75          elif doe_key=='cc':
76              assert 'centers' in doe_spec.keys()
77              assert 'alpha' in doe_spec.keys()
78              assert 'face' in doe_spec.keys()
```

```
79              num_fac = len(self.facs)
80              assert num_fac > 0
81              templist = doe_spec["centers"]
82              assert len(templist)==2
83              centers = tuple(templist)
84              alpha = doe_spec["alpha"]
85              assert (alpha=='orthogonal') or (alpha=='o') or (alpha=='
                    rotatable') or (alpha=='r')
86              face = doe_spec["face"]
87              assert (face=='circumscribed') or (face=='ccc') or (face=='
                    inscribed') or (face=='cci') or (face=='faced') or (face
                    =='ccf')
88              if face=='ccf':
89                  assert (alpha=='orthogonal') or (alpha=='o')
90              self.doe = DOE_MAPPER[doe_key](num_fac, centers, alpha, face
                    )
91          else:
92              print("Unsupported␣DoE␣Key␣in␣DoE␣Specification")
93          assert self.doe is not None
94
95      def get_factor_values(self):
96          assert self.doe is not None
97          assert len(self.facs) > 0
98          shape_t1 = self.doe.shape
99          outs = np.zeros(shape_t1)
100         for x in range(shape_t1[1]):
101             outs[:,x] = self.facs[x].denormalize(self.doe[:,x])
102         return outs.tolist()
103
104
105 def format_output_text(headers,data):
106     output = ''
107     outlist = []
```

```
108     hlist = []
109     for ndx in range(len(headers)):
110         hlist.append(headers[ndx])
111         if ndx<(len(headers)-1):
112             output = output + headers[ndx] + ','
113         else:
114             output = output + headers[ndx] + '\n'
115     outlist.append(list(hlist))
116     for ndx in range(len(data)):
117         thisrow = data[ndx]
118         outlist.append(list(thisrow))
119         for mdx in range(len(thisrow)):
120             itm = thisrow[mdx]
121             if mdx<(len(thisrow)-1):
122                 output = output + '{}'.format(itm) + ','
123             else:
124                 output = output + '{}'.format(itm)
125         if ndx<(len(data)-1):
126             output = output + '\n'
127     return output, list(outlist)
128
129
130 def main(arg1, arg2):
131     # arg1 should be text representing the DoE Specification in JSON
132     # arg2 should be text representing the Factor Specification in JSON
133     DoE_Spec = json.loads(arg1)
134     Factor_Spec = json.loads(arg2)
135     #print(Factor_Spec)
136     this_experiment = Experiment(Factor_Spec, 'auto')
137     this_experiment.build_factors()
138     this_experiment.build_doe(DoE_Spec)
139     otext, olist = format_output_text([f.name for f in this_experiment.
            facs], list(this_experiment.get_factor_values()))
```

```
140     return otext, olist
141
142
143 if __name__ == '__main__':
144     #print('Hello from Python')
145     aname = sys.argv[4]
146     fp = sys.argv[3]
147     doe = sys.argv[2]
148     fac = sys.argv[1]
149     otext, olist = main(doe, fac)
150     # sys.stdout.write(json.dumps(otext))
151     # sys.stdout.flush()
152     with open(path.join(fp,'{}_doe.csv'.format(aname)), 'w', newline='')
            as cf:
153         cwriter = csv.writer(cf)
154         for row in olist:
155             cwriter.writerow(row)
156     sys.stdout.write(path.join(fp,'{}_doe.csv'.format(aname)))
157     sys.stdout.flush()
158 #
```

APPENDIX B

FIGURES FOR WALWORTH CASES

Experiment 1 ran a simple DoE on the Walworth[42] model. The plotted results (using [109]) are given in this Appendix.

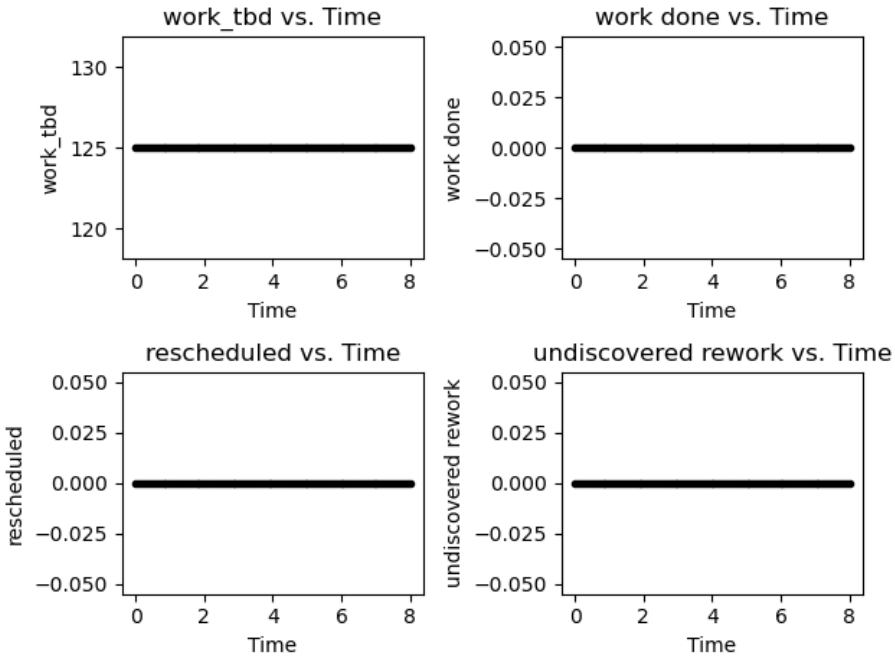


Figure B.1: Walworth Case 0000

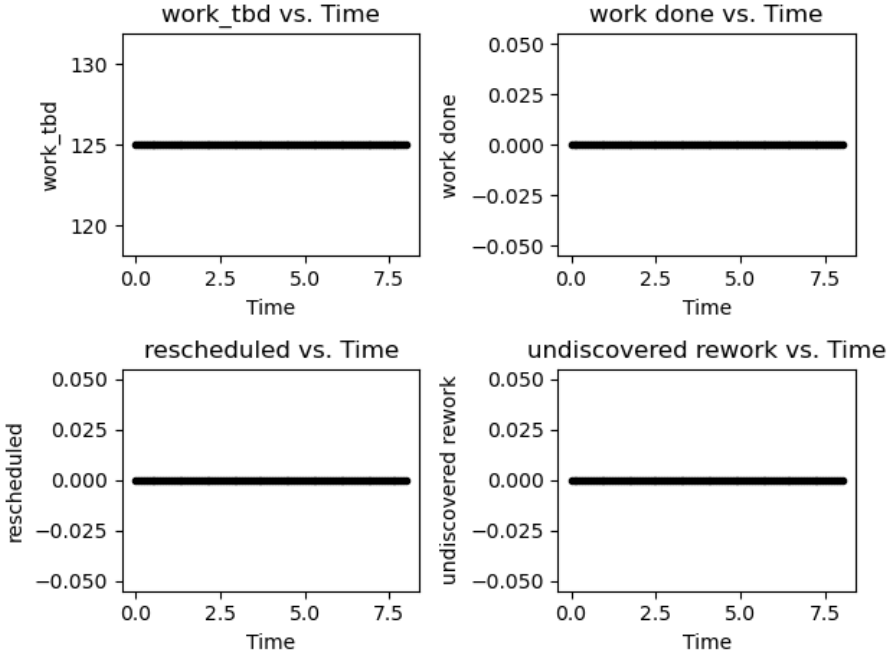


Figure B.2: Walworth Case 0001

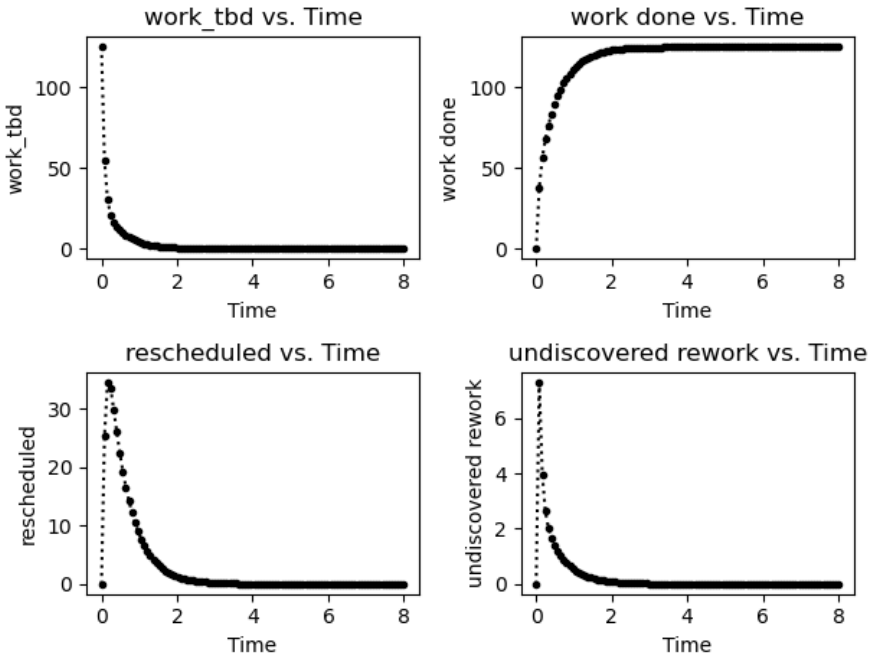


Figure B.3: Walworth Case 0002

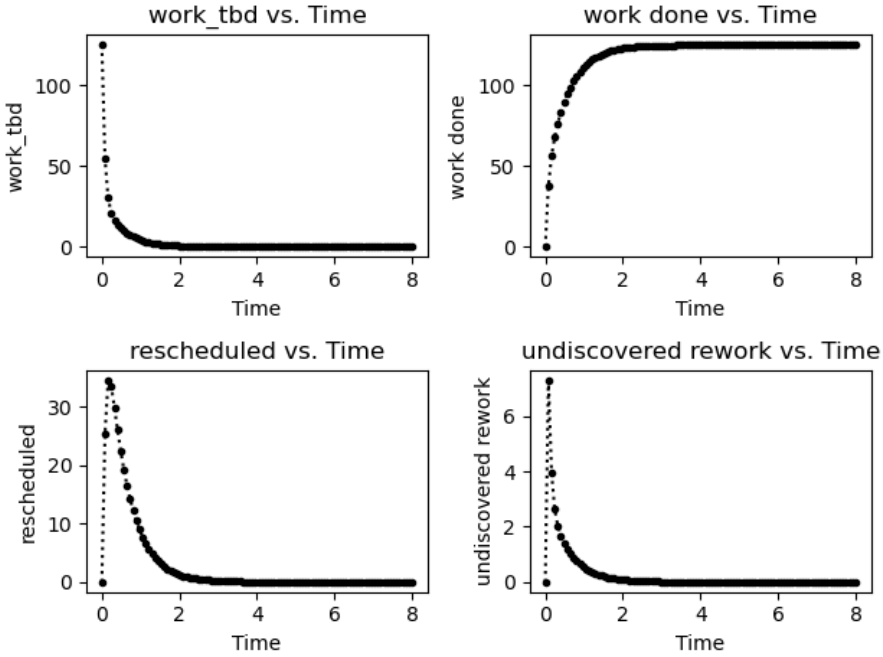


Figure B.4: Walworth Case 0003

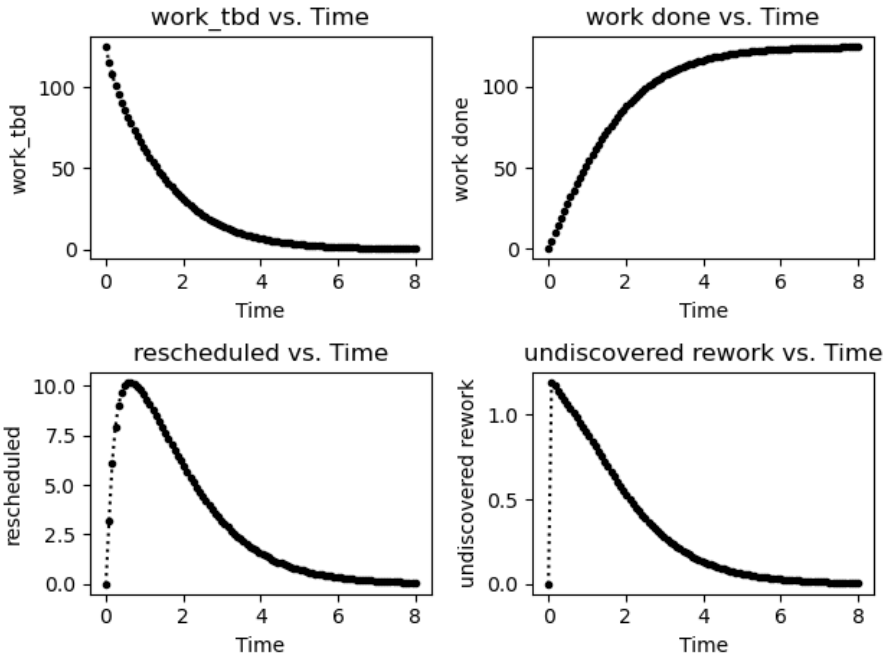


Figure B.5: Walworth Case 0004

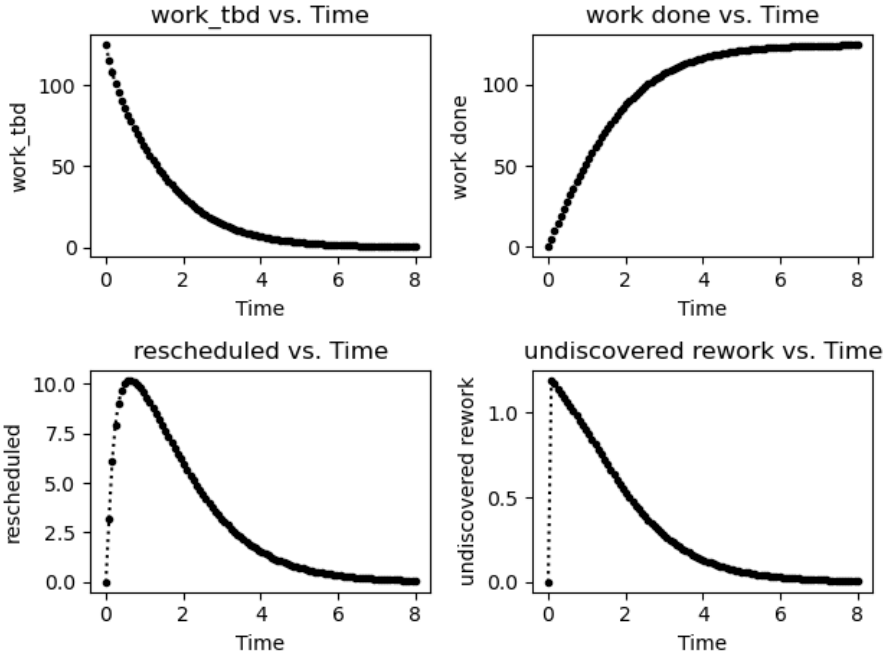


Figure B.6: Walworth Case 0005

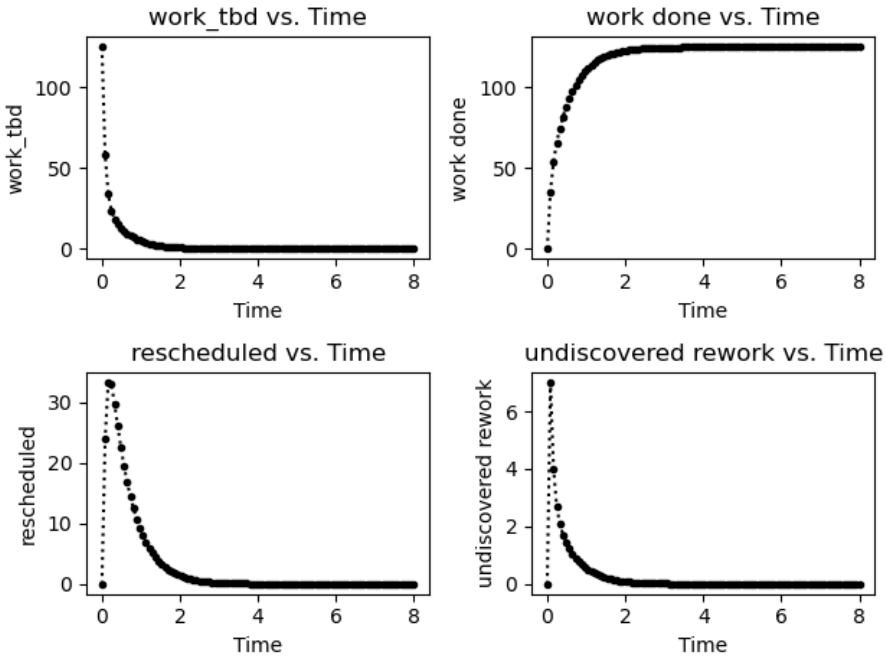


Figure B.7: Walworth Case 0006

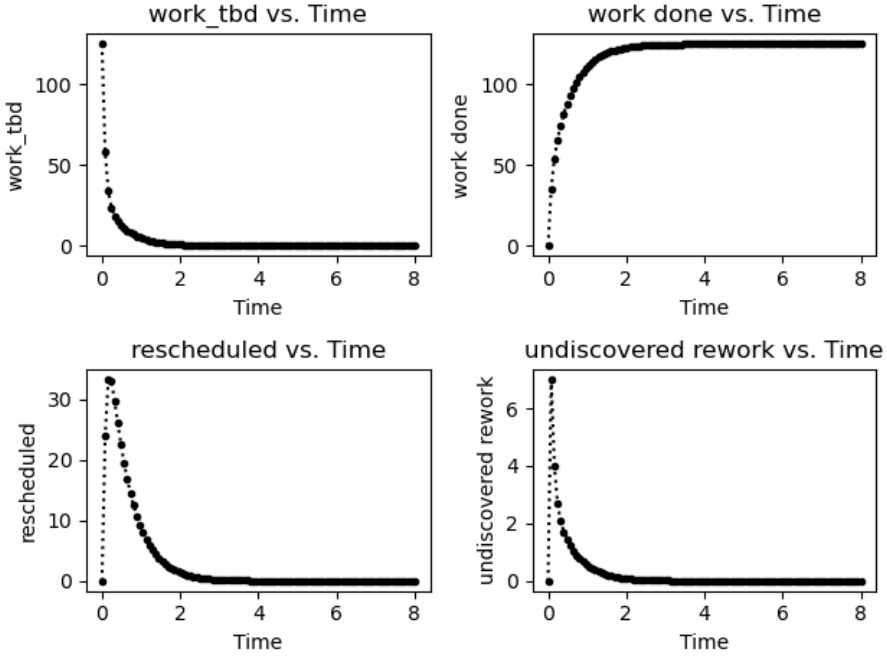


Figure B.8: Walworth Case 0007

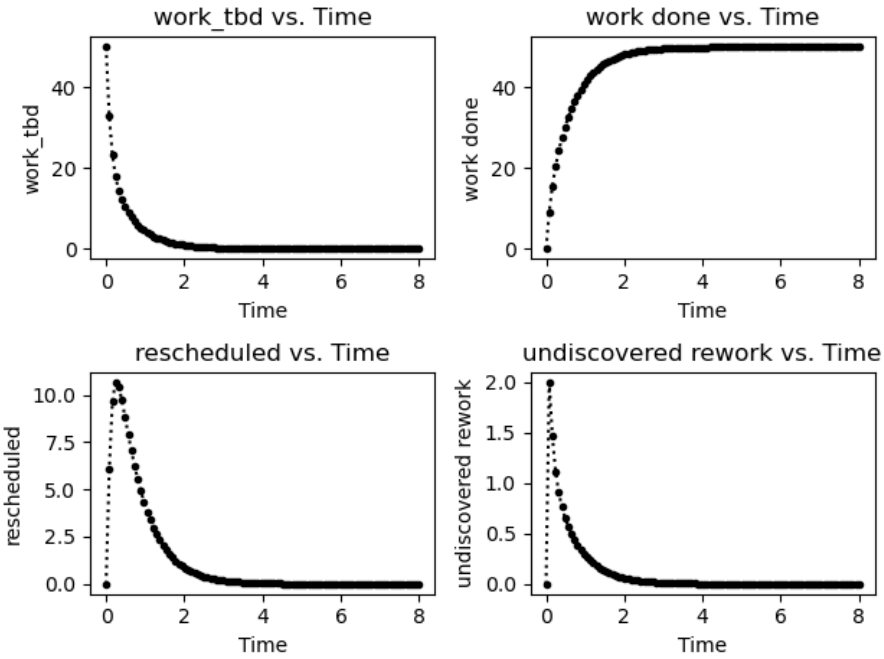


Figure B.9: Walworth Case 0008

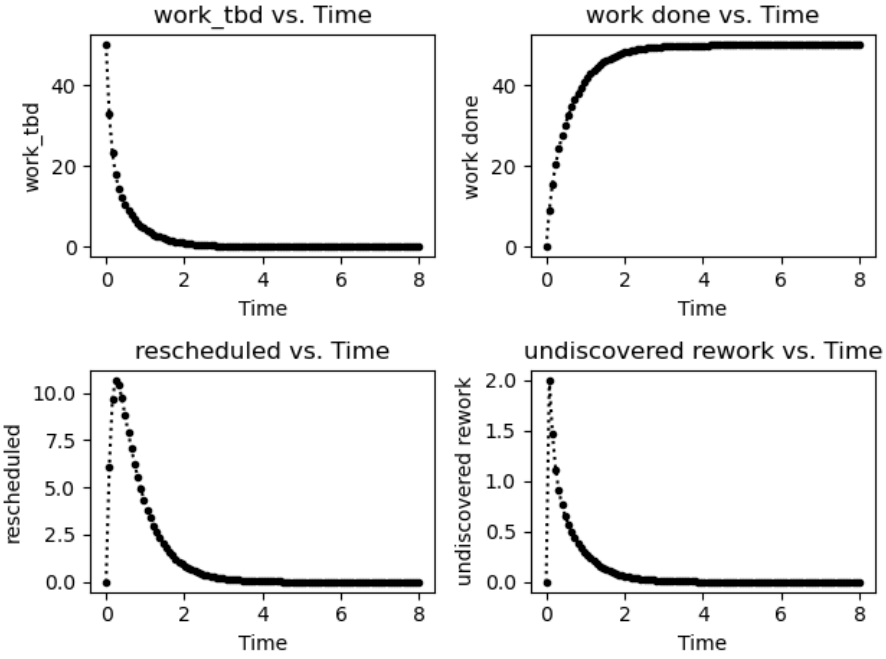


Figure B.10: Walworth Case 0009

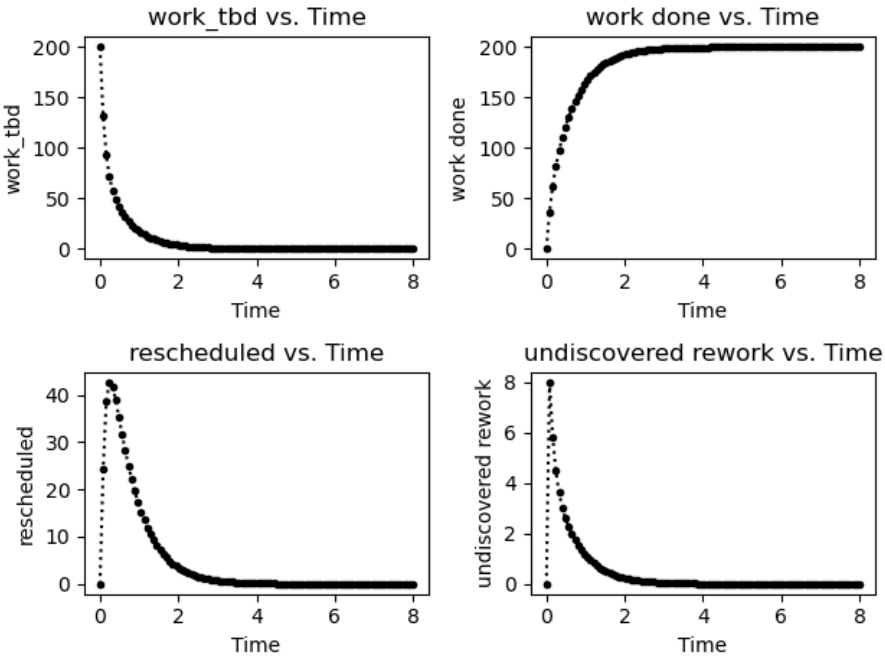


Figure B.11: Walworth Case 0010

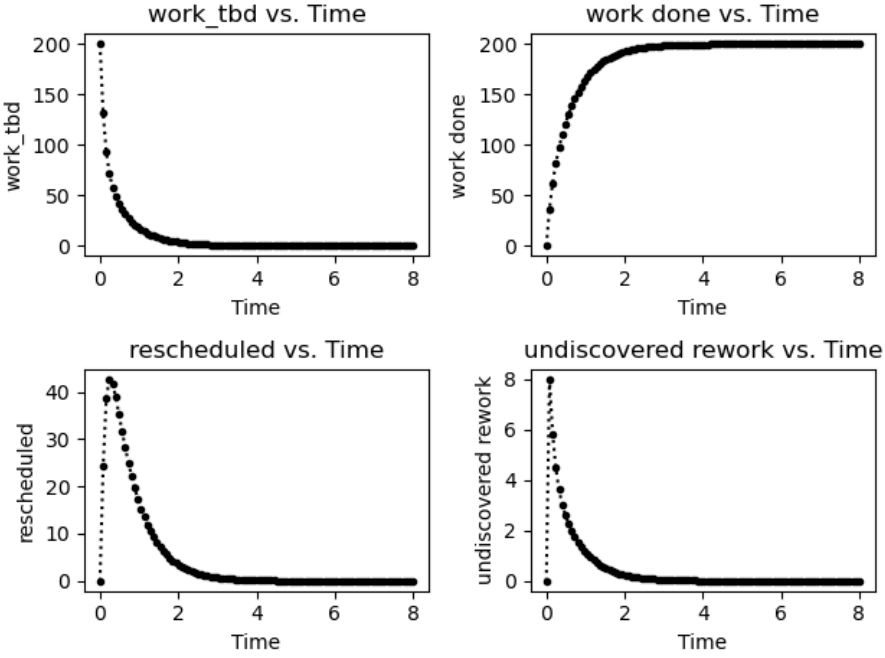


Figure B.12: Walworth Case 0011

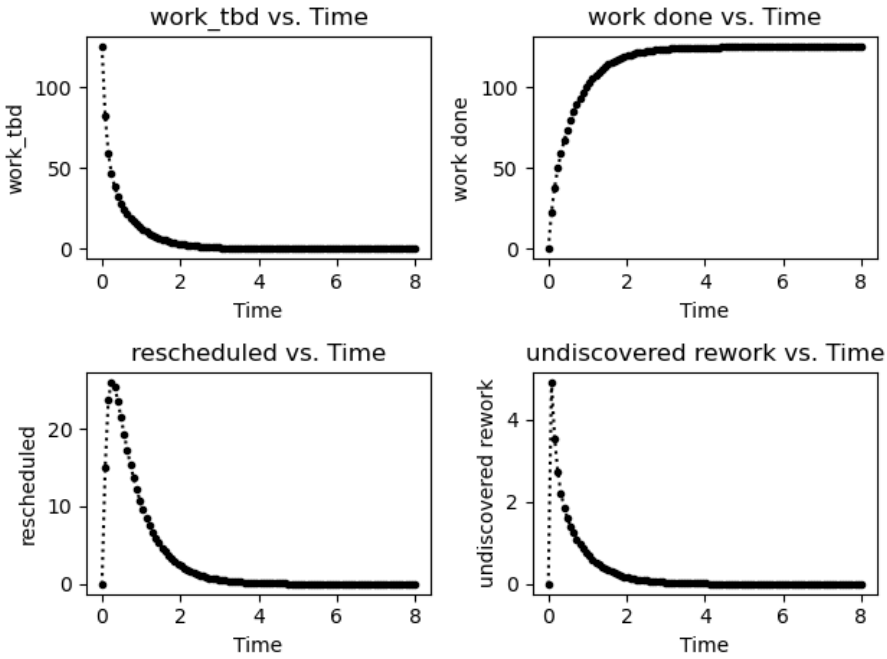


Figure B.13: Walworth Case 0012

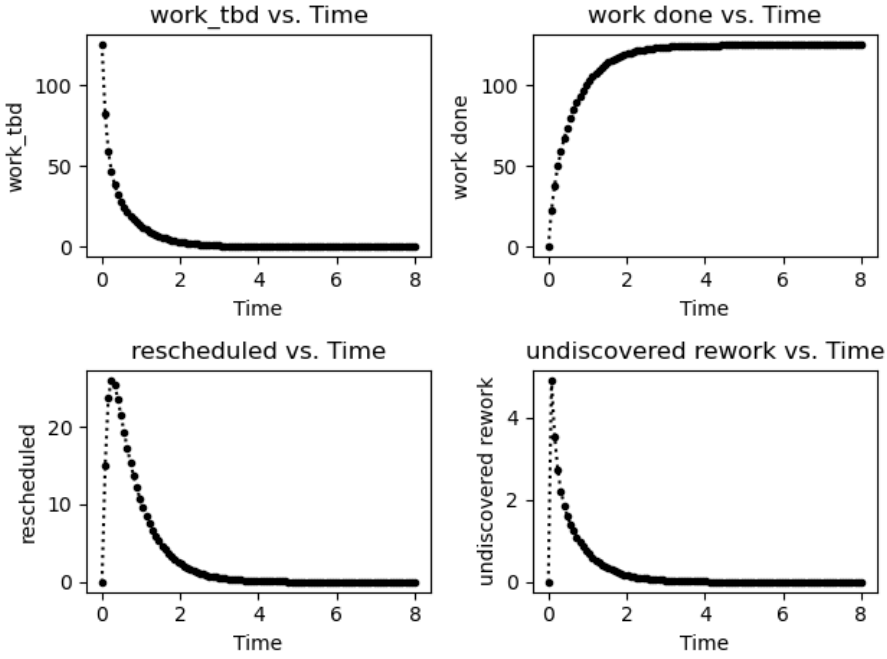


Figure B.14: Walworth Case 0013

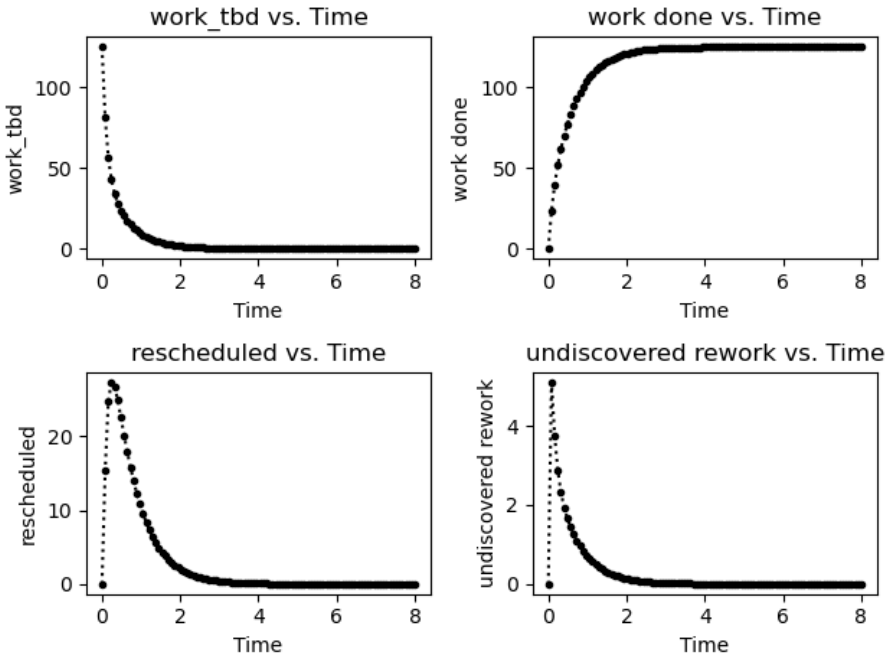


Figure B.15: Walworth Case 0014

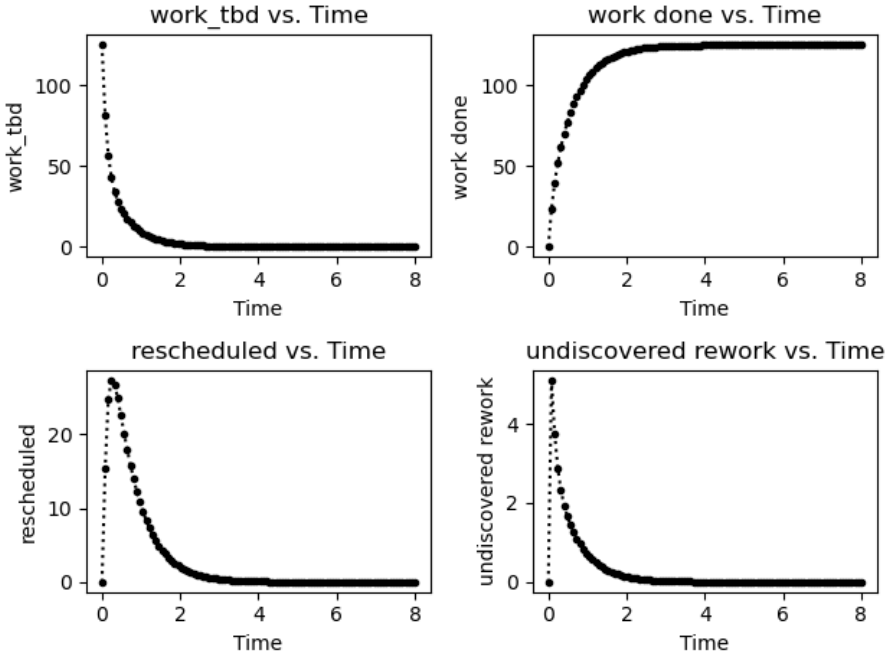


Figure B.16: Walworth Case 0015

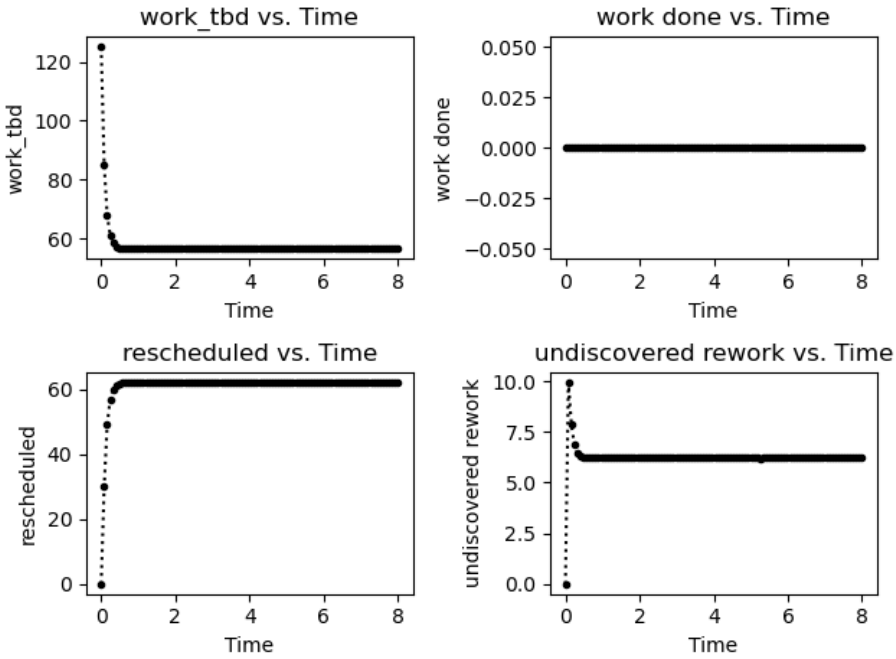


Figure B.17: Walworth Case 0016

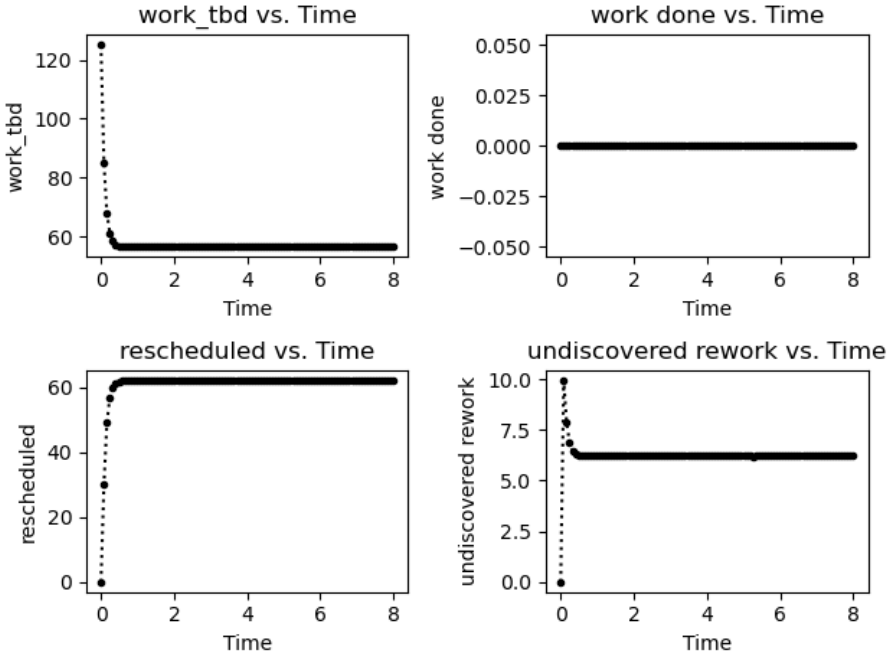


Figure B.18: Walworth Case 0017

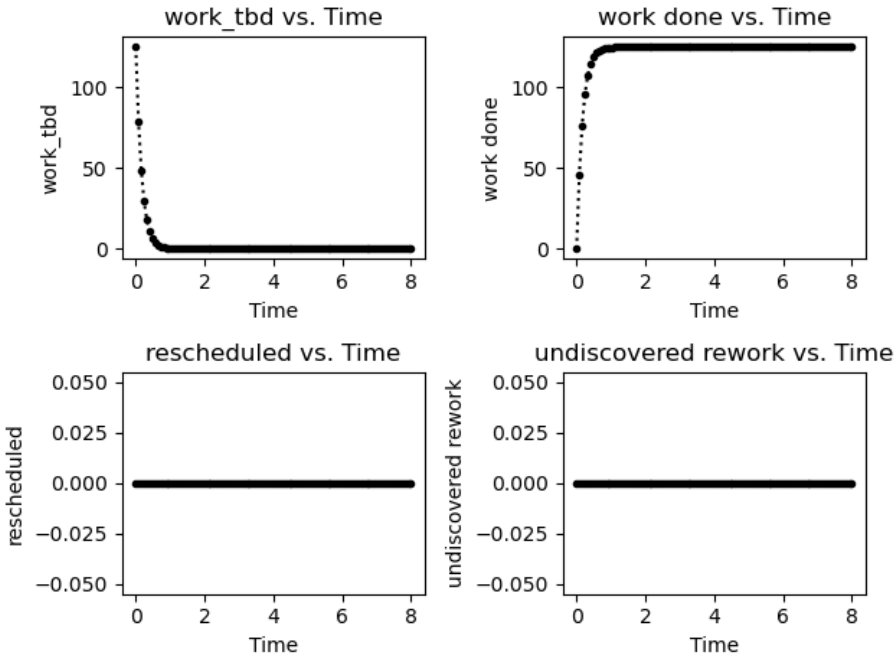


Figure B.19: Walworth Case 0018

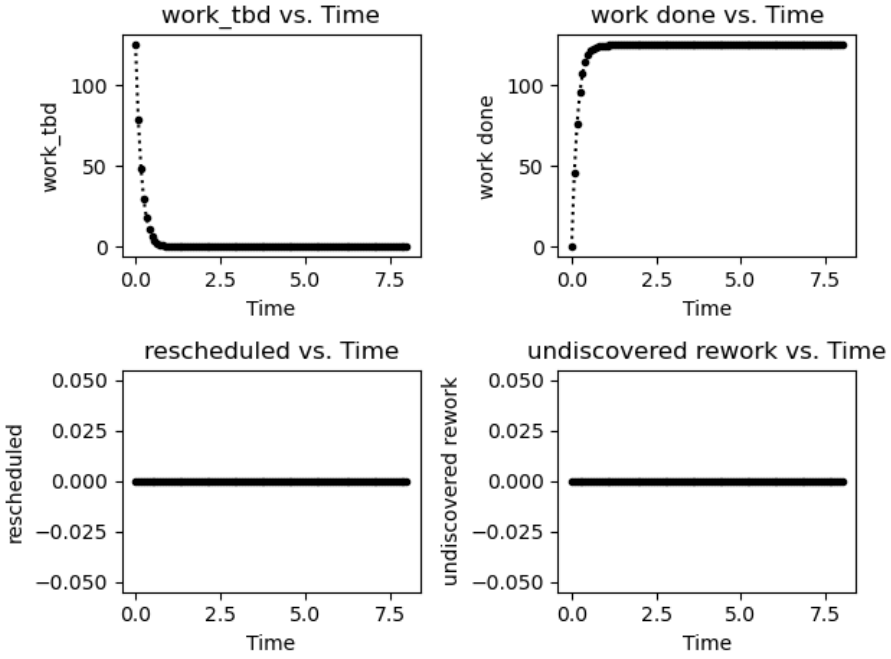


Figure B.20: Walworth Case 0019

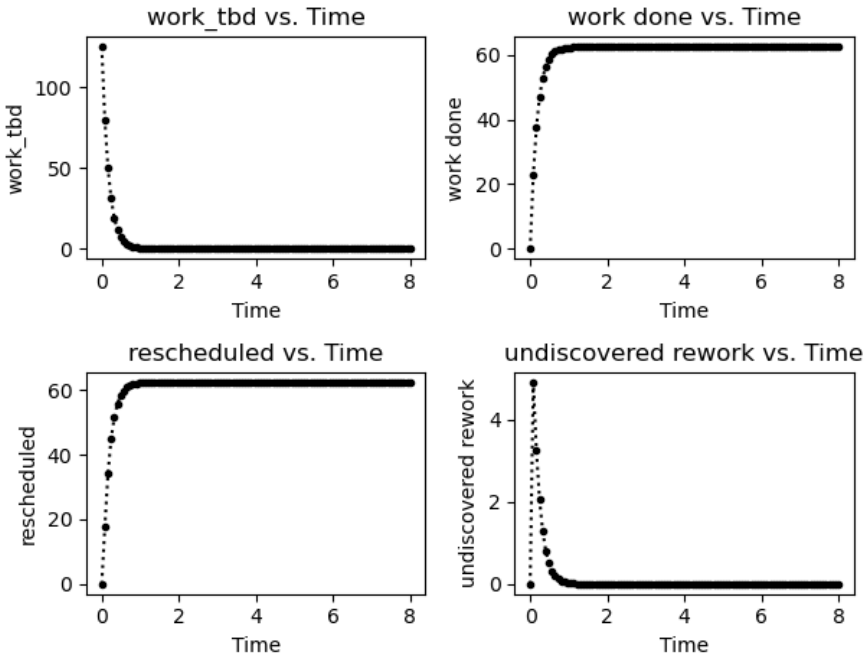


Figure B.21: Walworth Case 0020

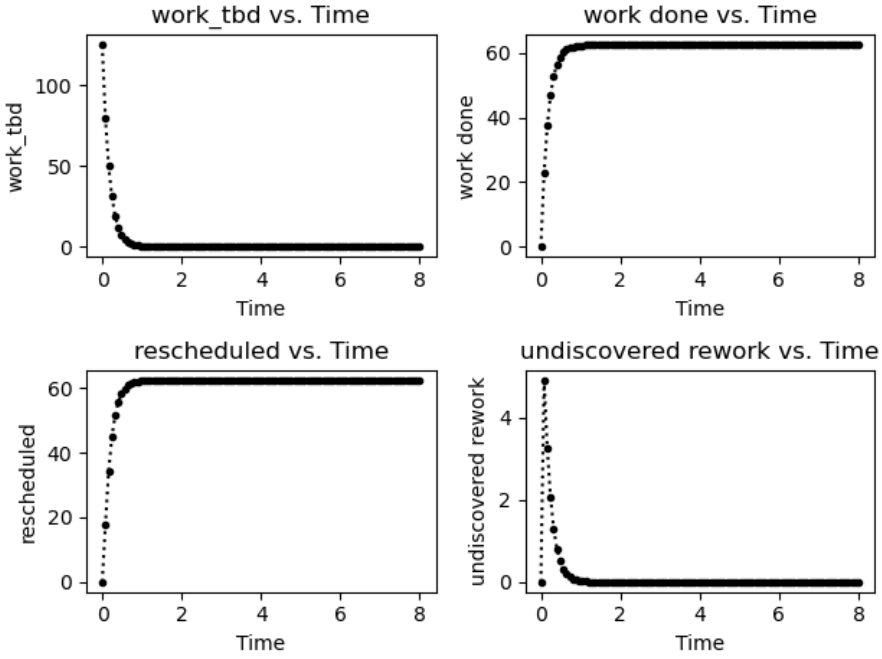


Figure B.22: Walworth Case 0021

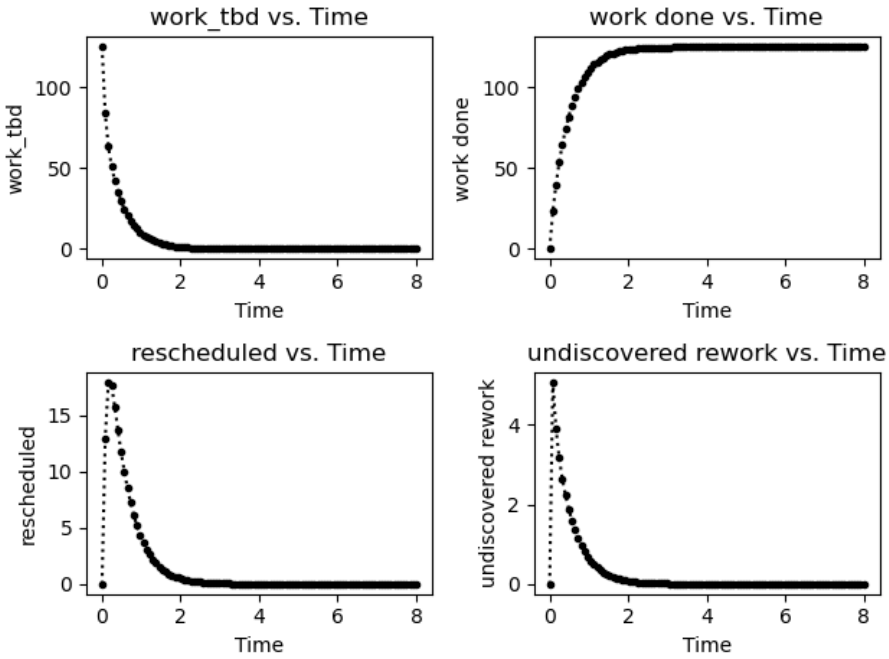


Figure B.23: Walworth Case 0022

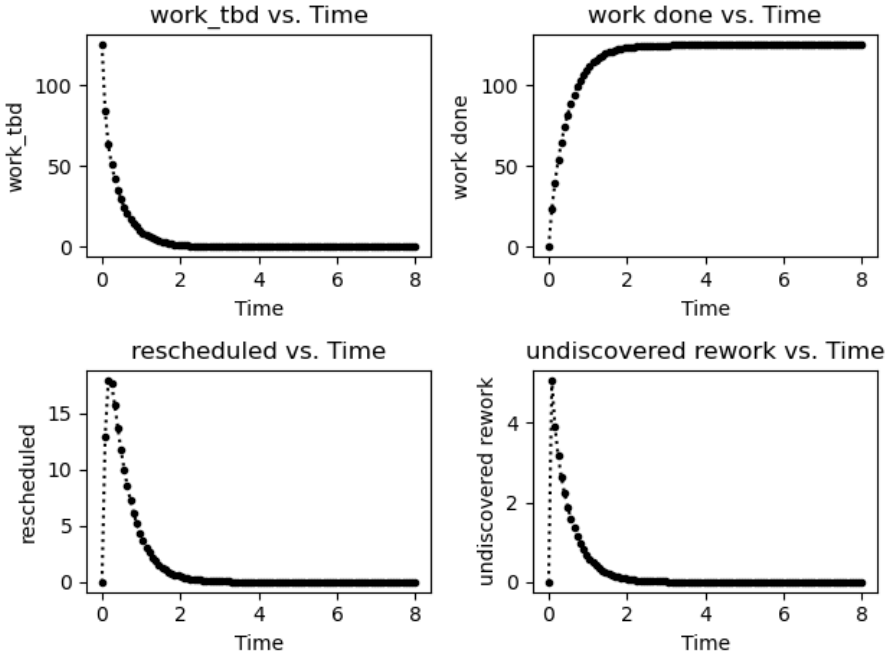


Figure B.24: Walworth Case 0023

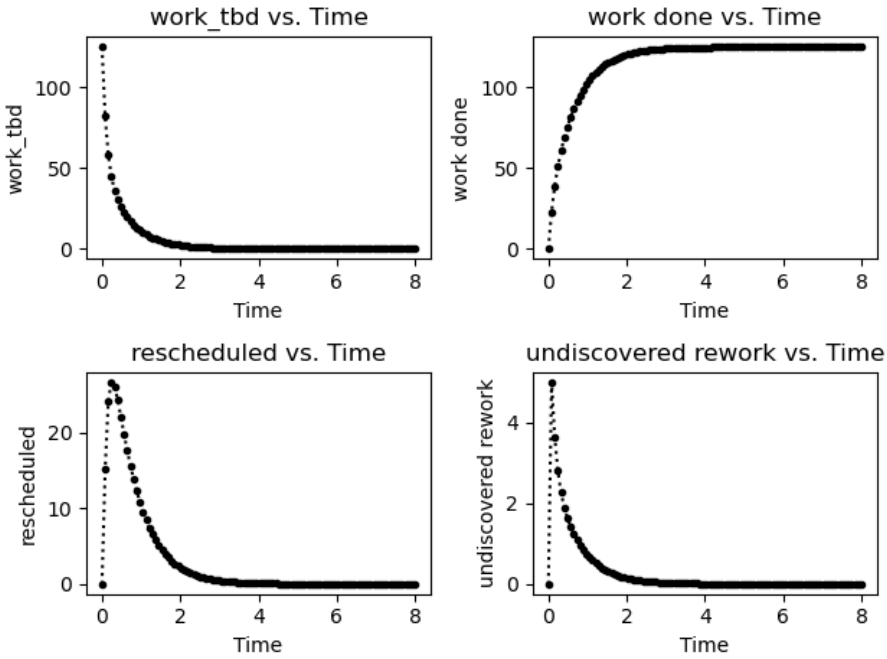


Figure B.25: Walworth Case 0024

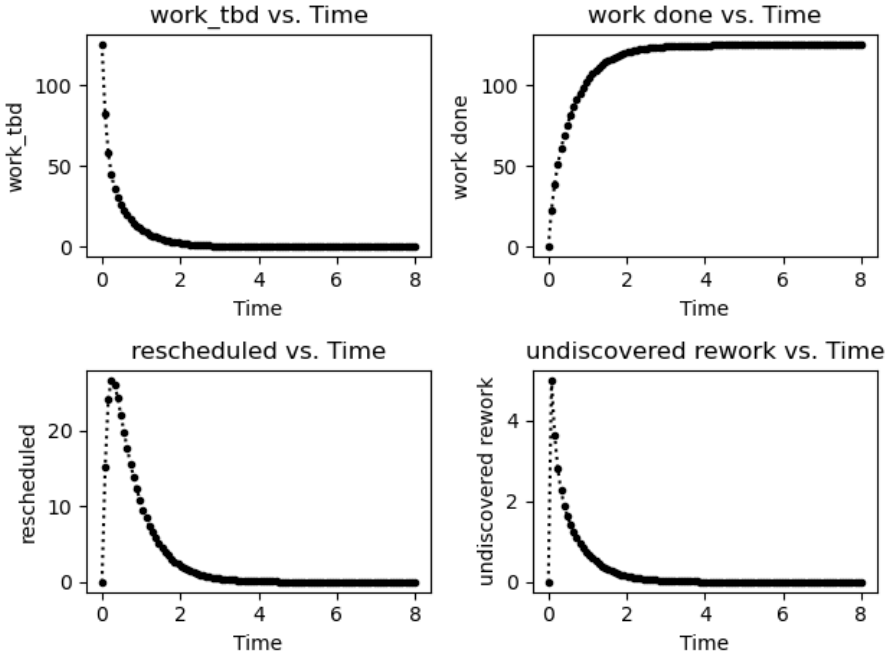


Figure B.26: Walworth Case 0025

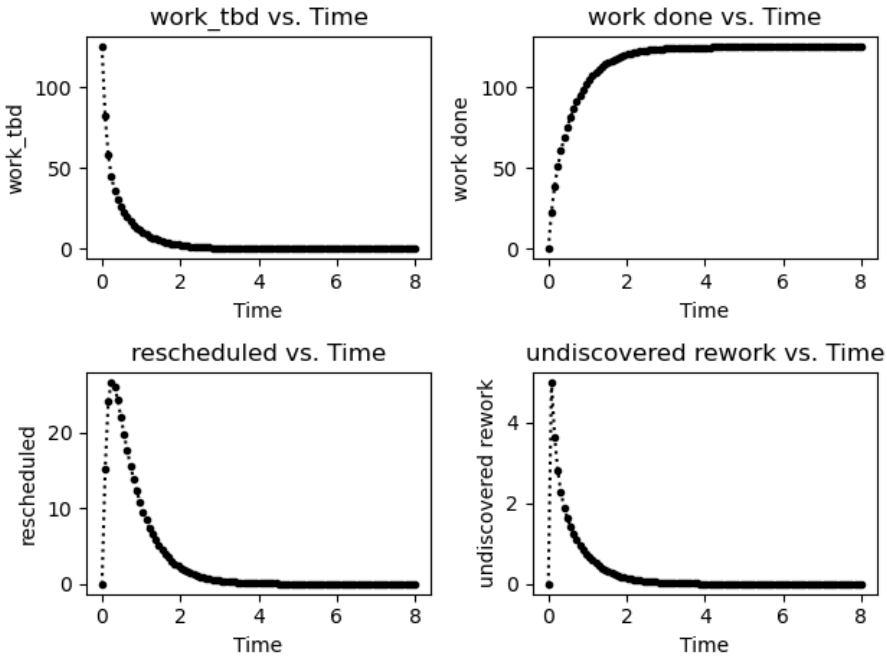


Figure B.27: Walworth Case 0026

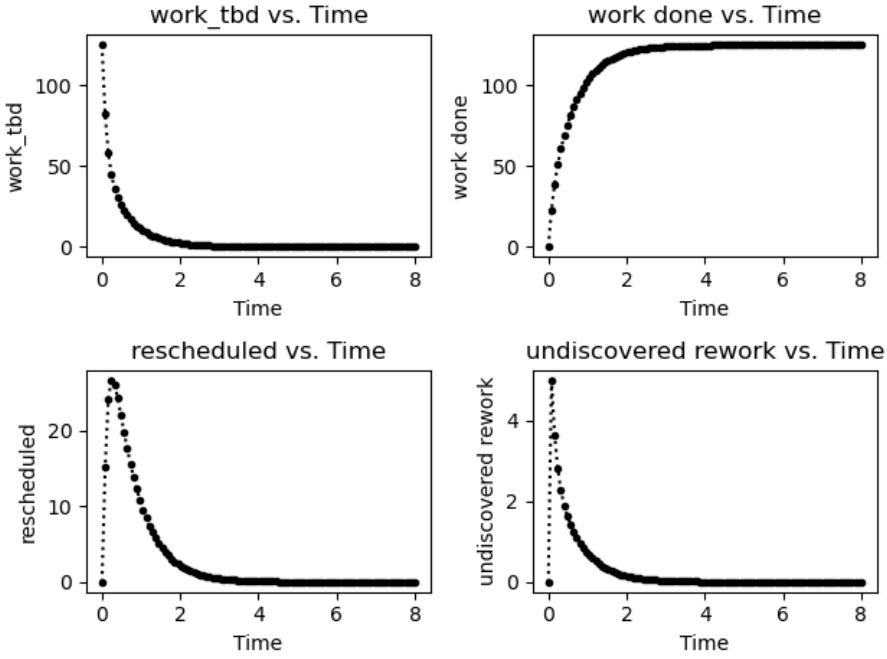


Figure B.28: Walworth Case 0027

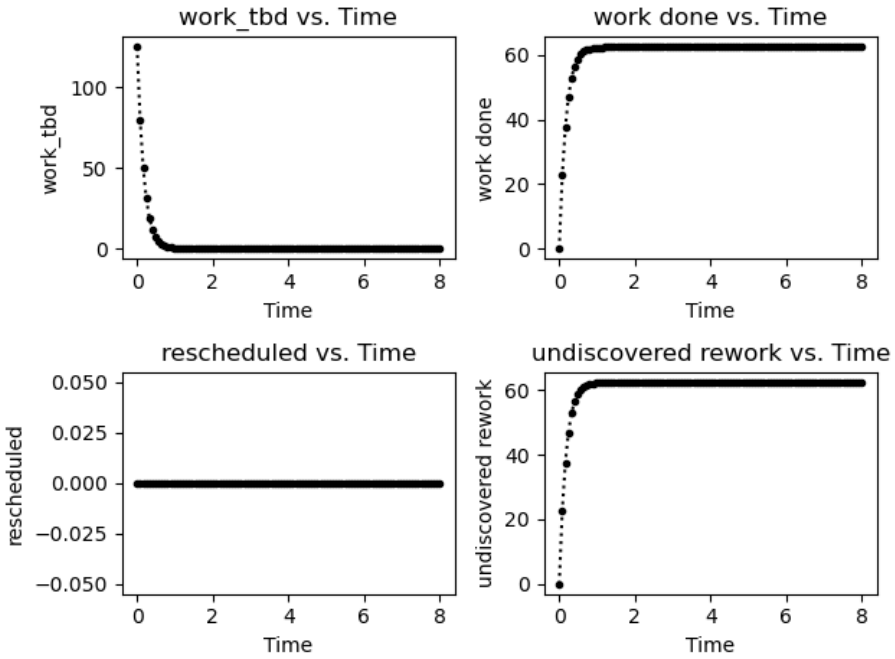


Figure B.29: Walworth Case 0028

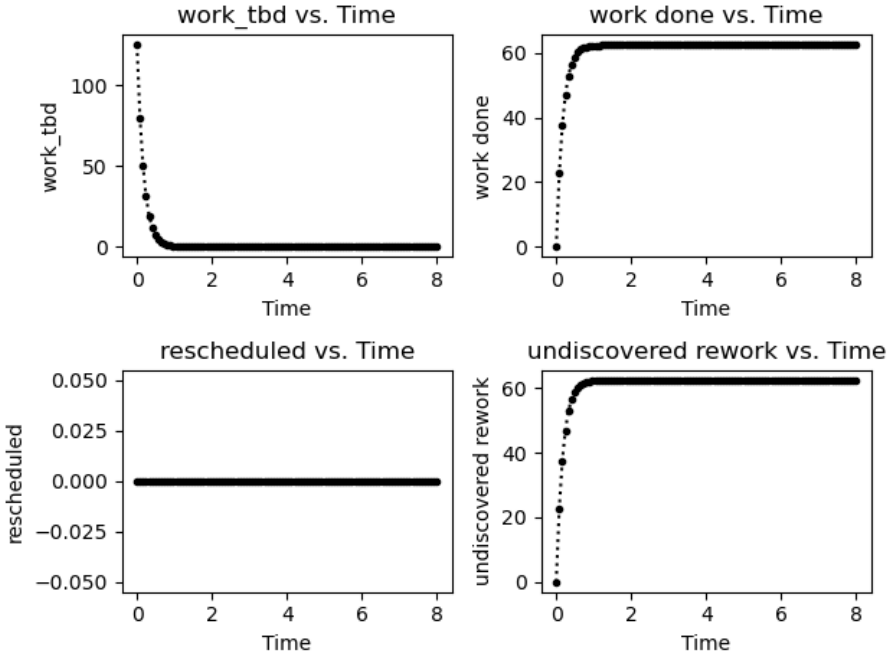


Figure B.30: Walworth Case 0029

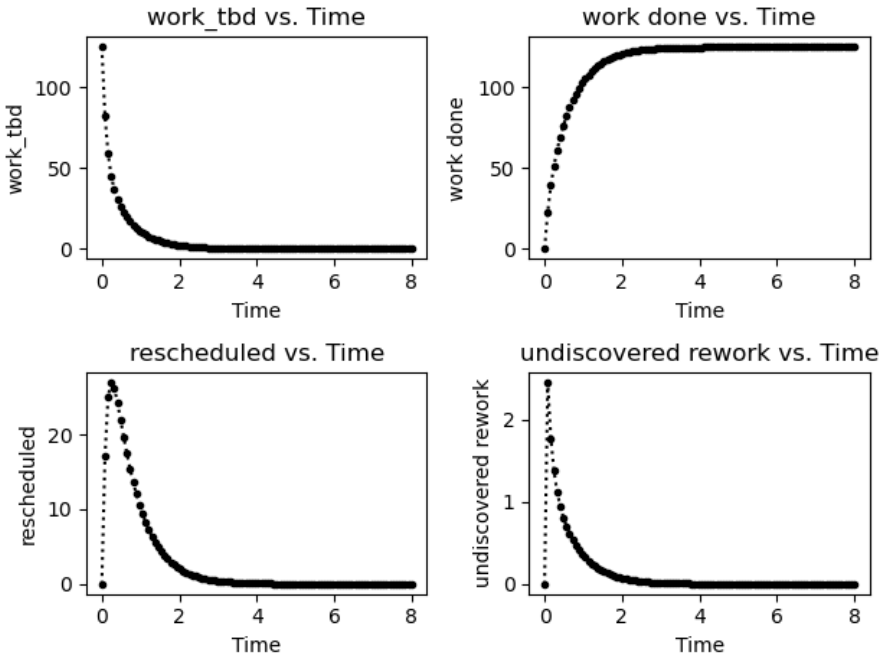


Figure B.31: Walworth Case 0030

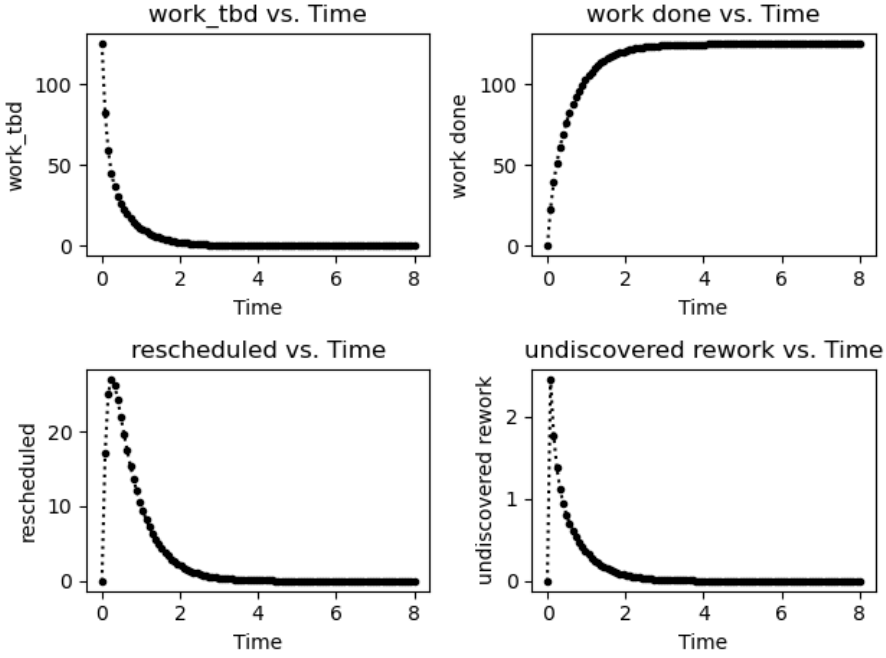


Figure B.32: Walworth Case 0031

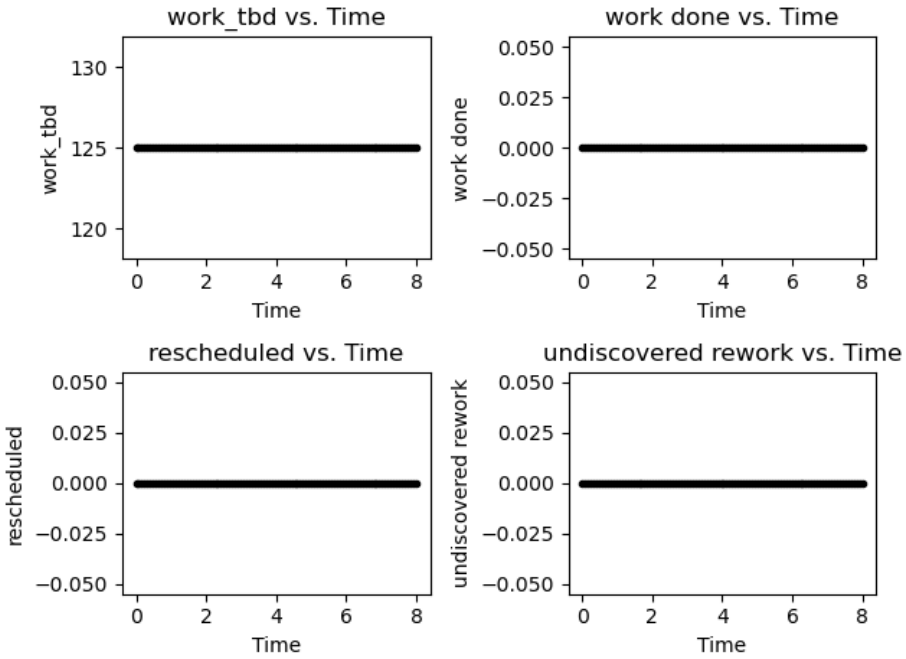


Figure B.33: Walworth Case 0032

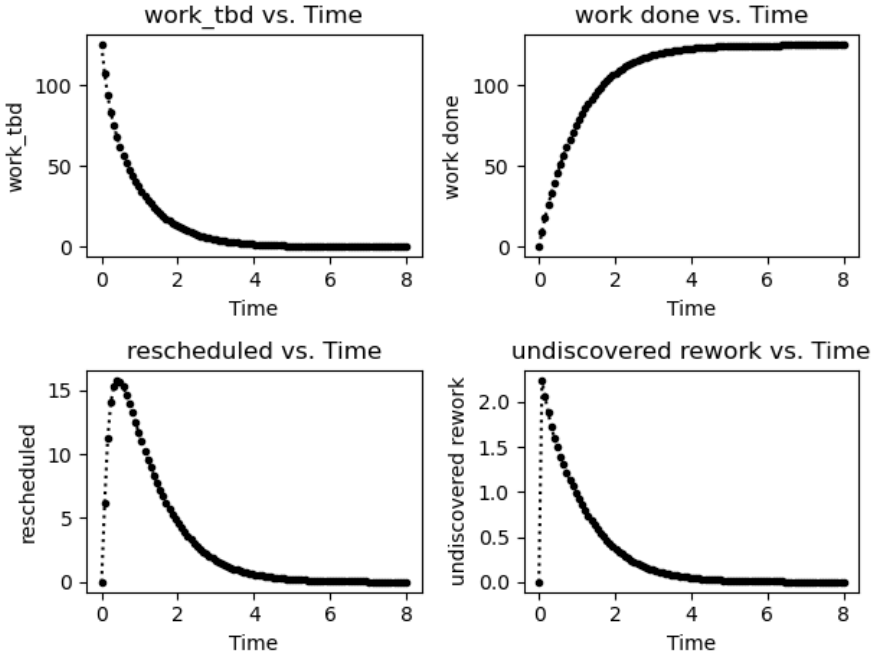


Figure B.34: Walworth Case 0033

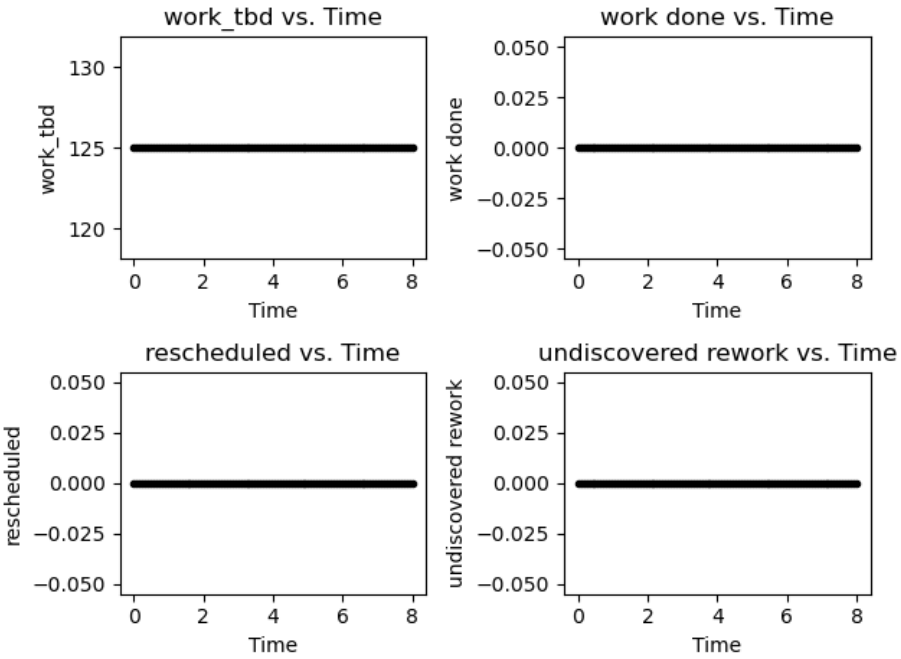


Figure B.35: Walworth Case 0034

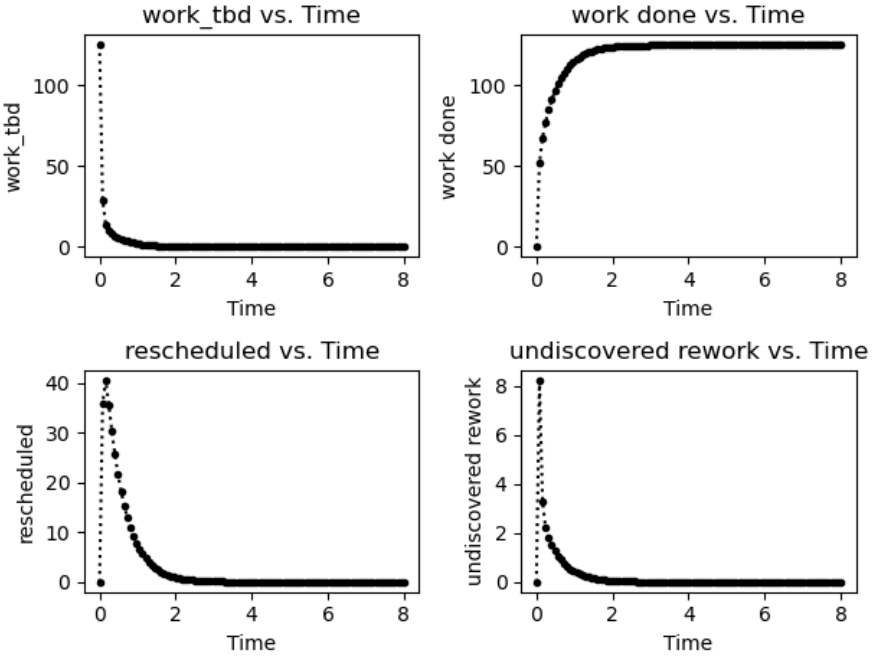


Figure B.36: Walworth Case 0035

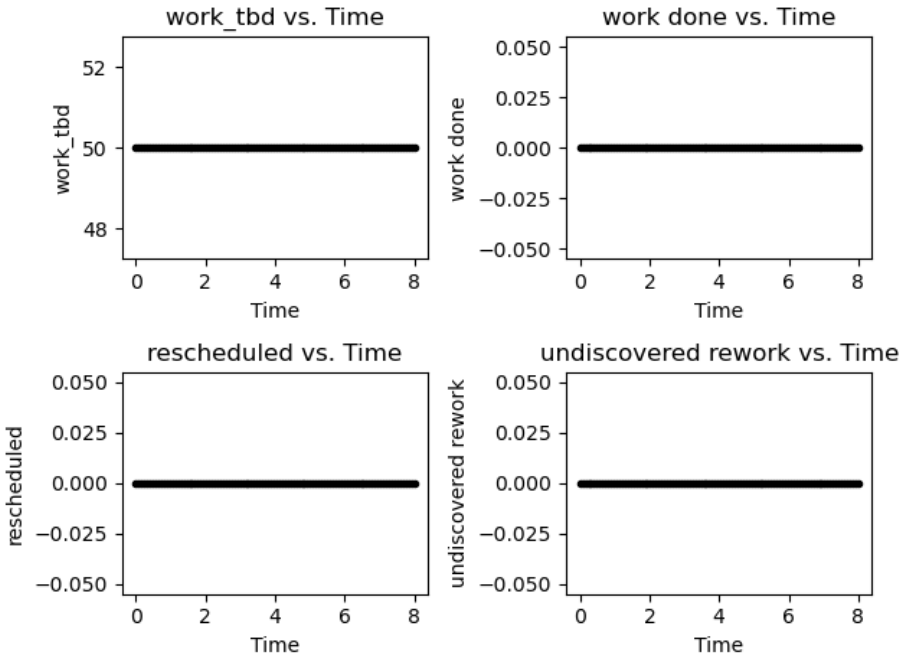


Figure B.37: Walworth Case 0036

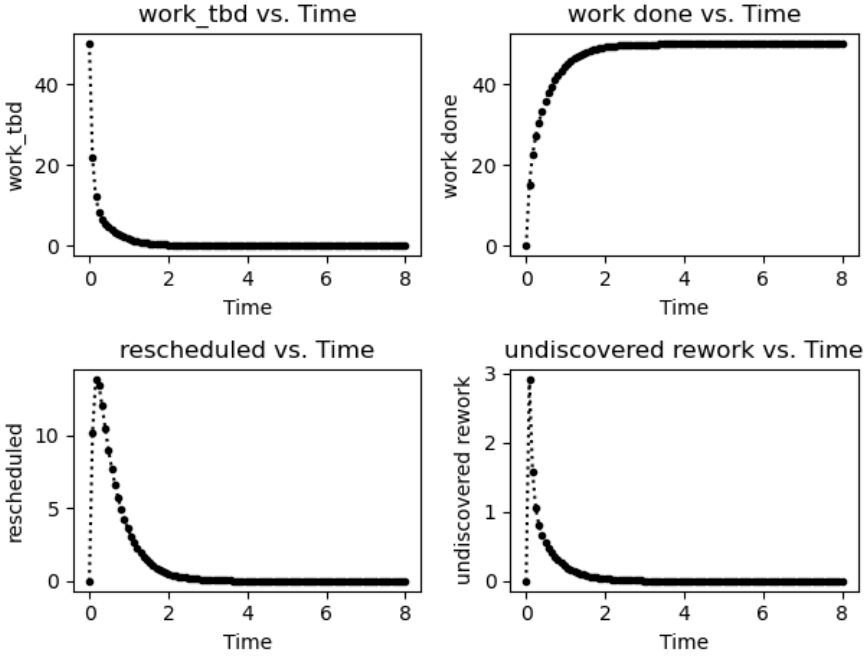


Figure B.38: Walworth Case 0037

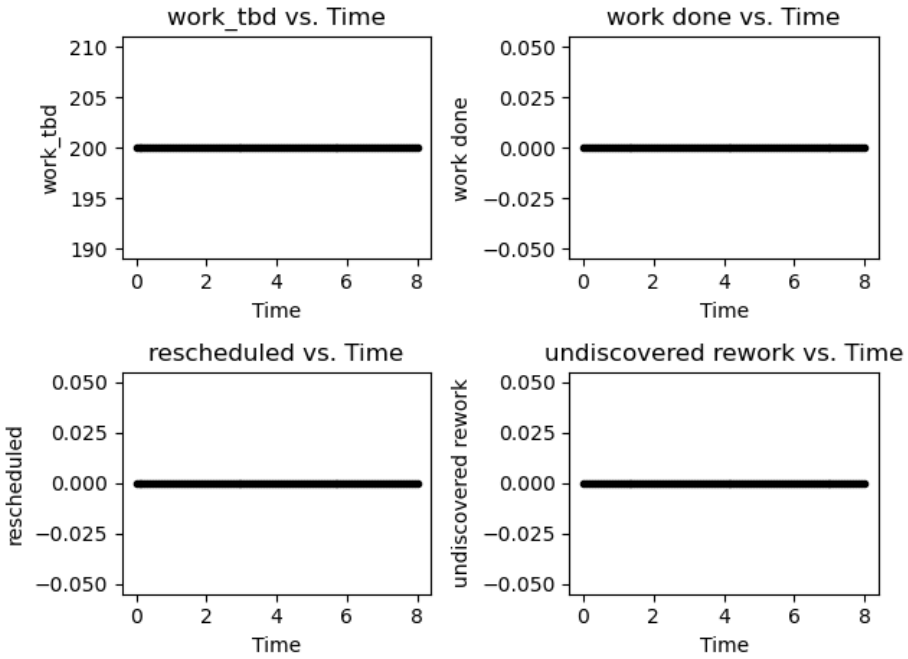


Figure B.39: Walworth Case 0038

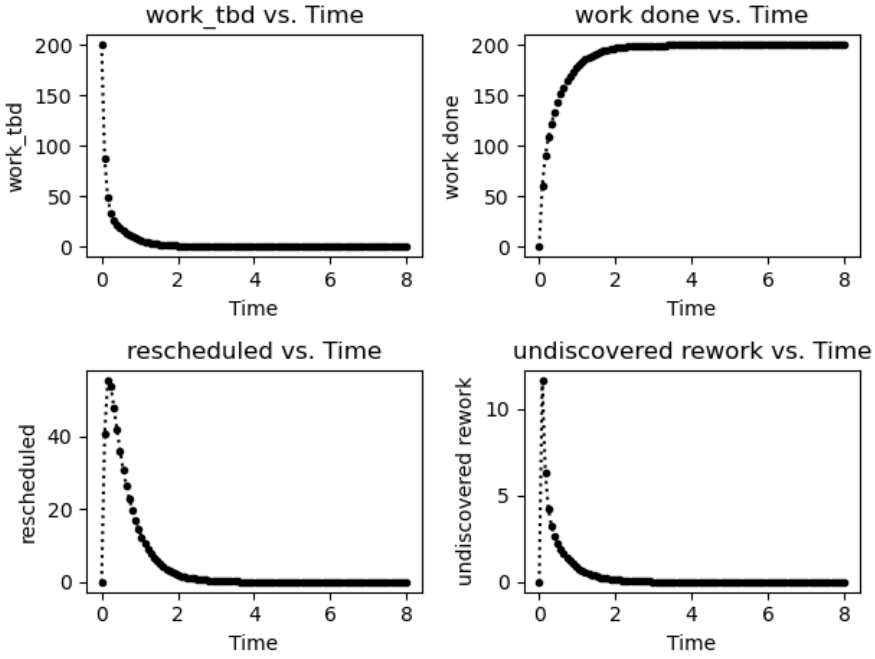


Figure B.40: Walworth Case 0039

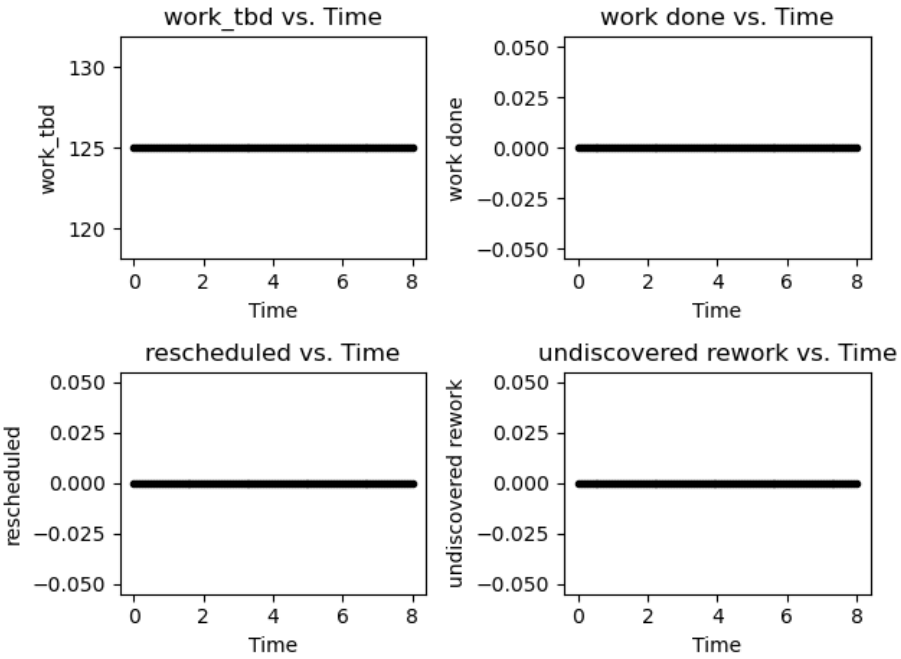


Figure B.41: Walworth Case 0040

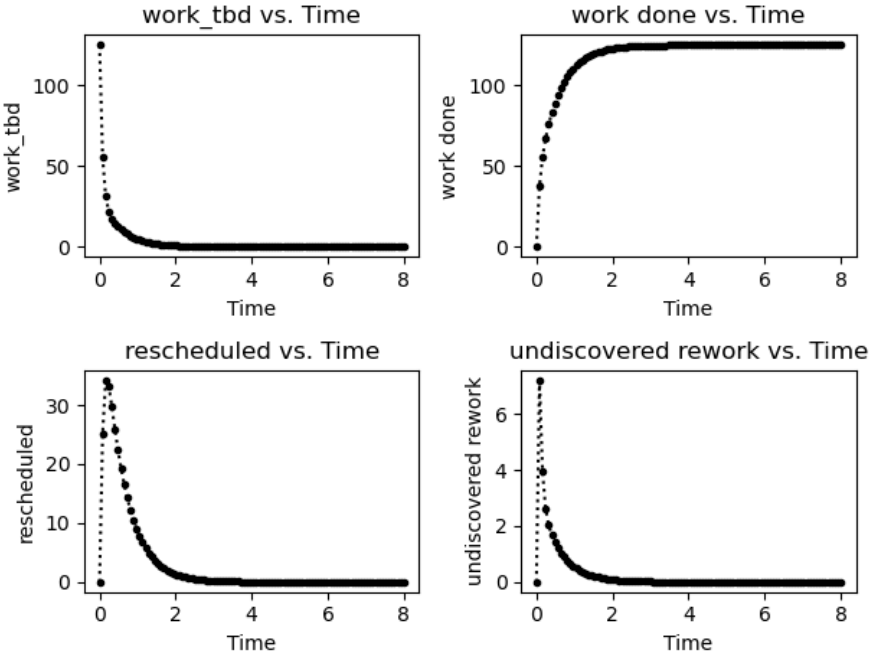


Figure B.42: Walworth Case 0041

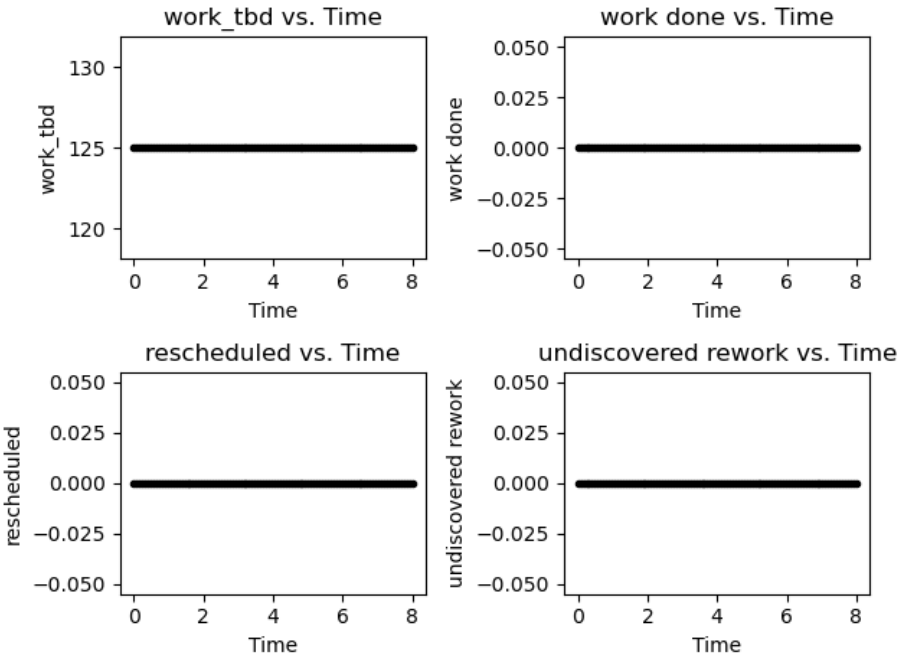


Figure B.43: Walworth Case 0042

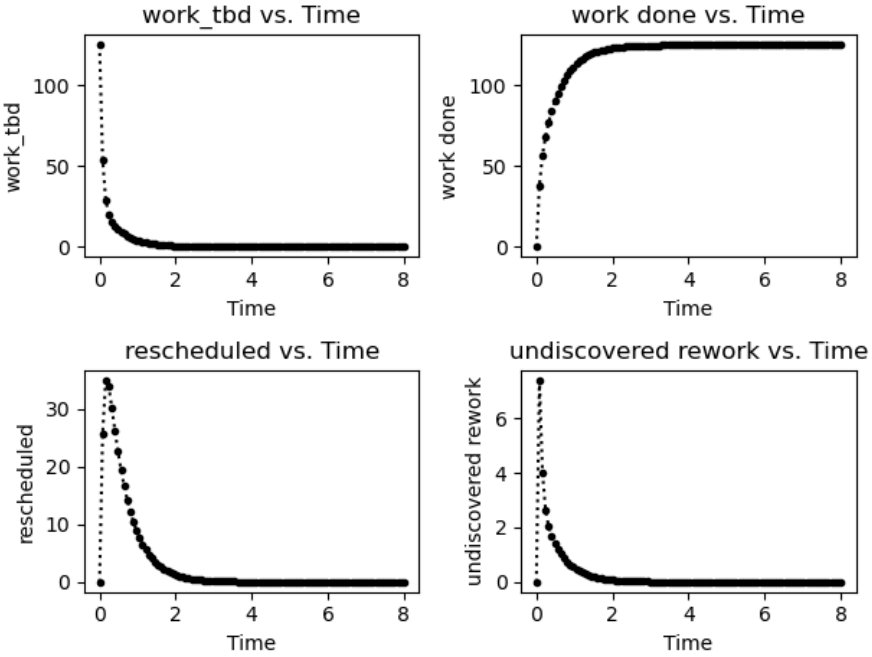


Figure B.44: Walworth Case 0043

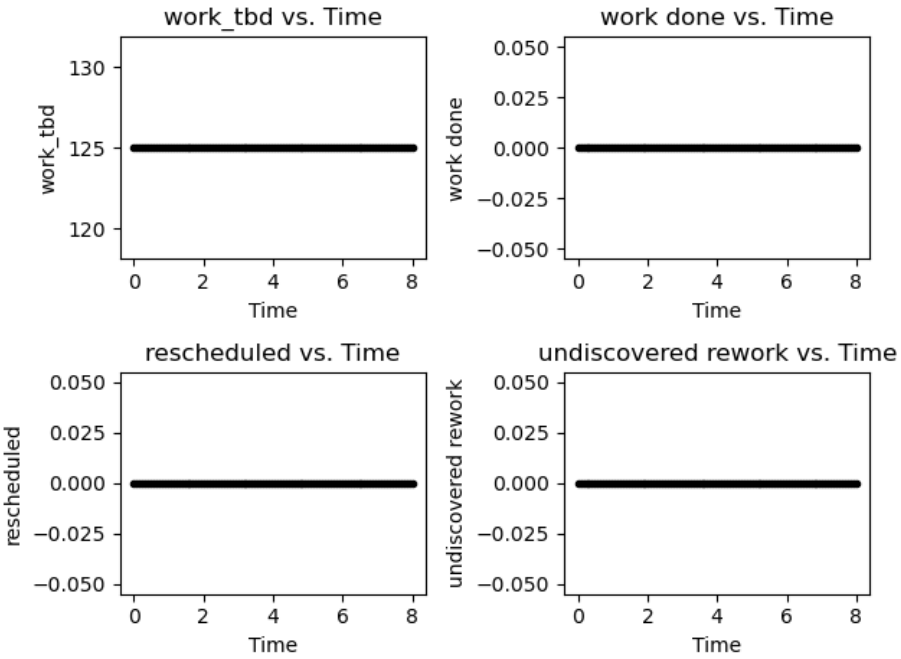


Figure B.45: Walworth Case 0044

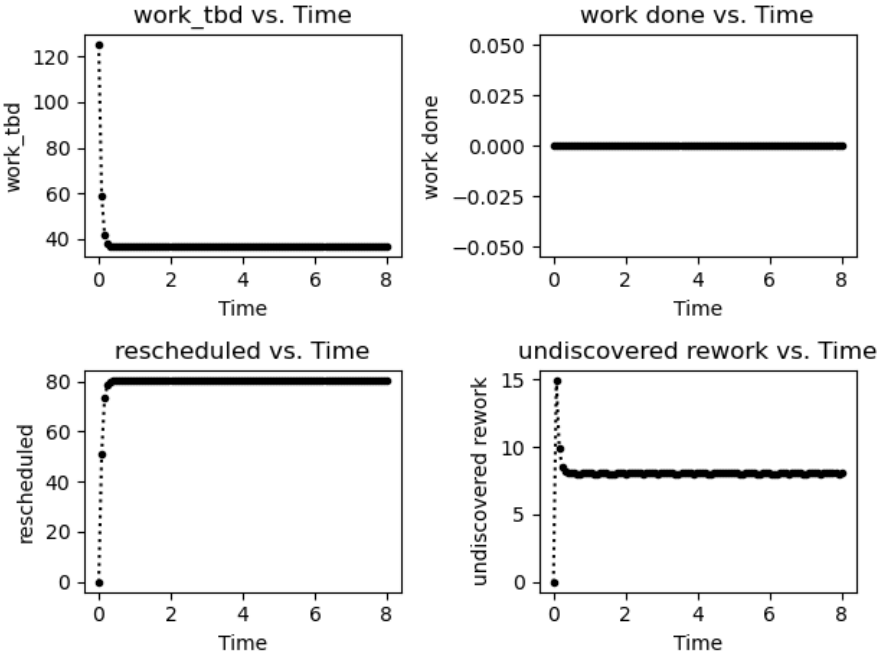


Figure B.46: Walworth Case 0045

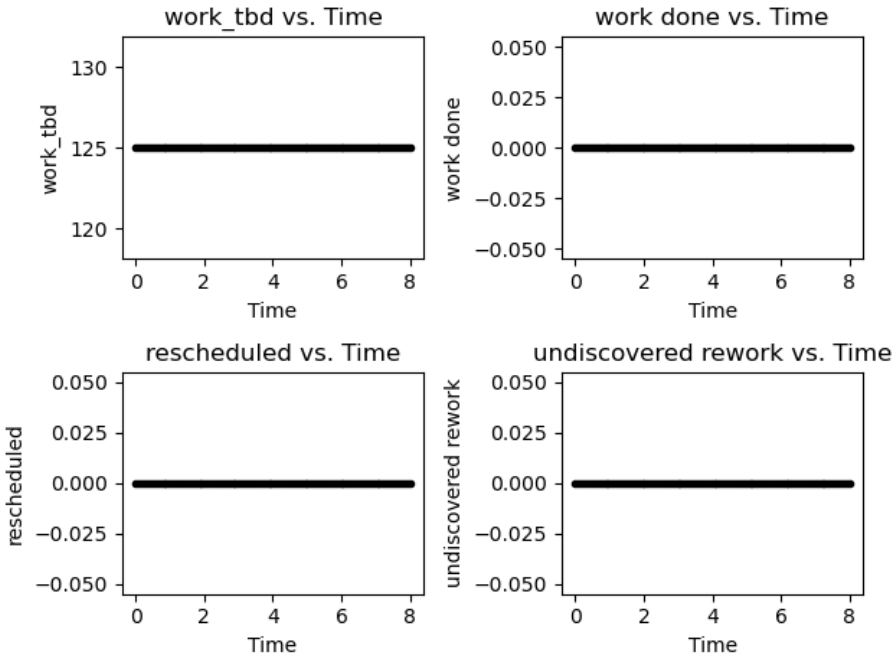


Figure B.47: Walworth Case 0046

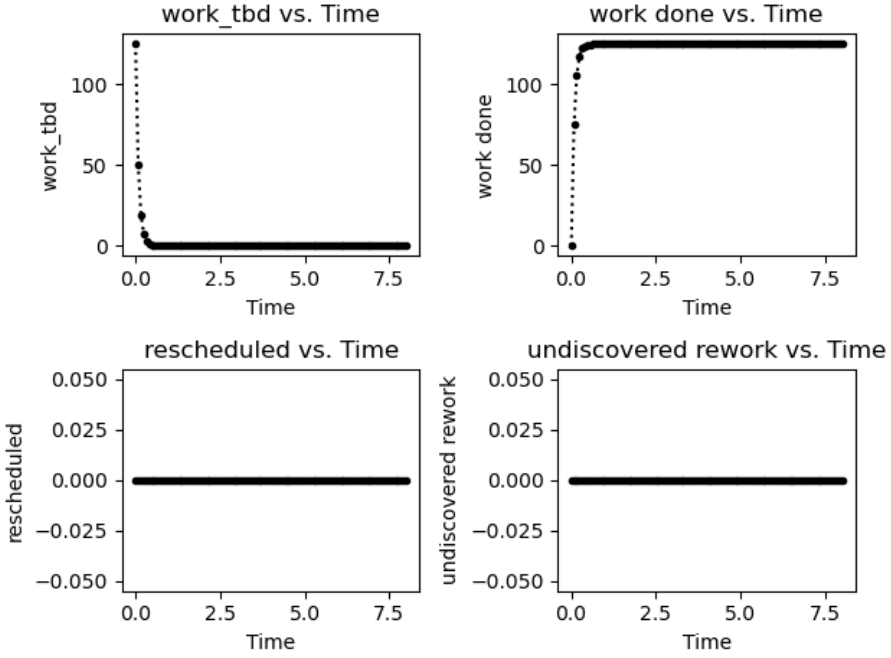


Figure B.48: Walworth Case 0047

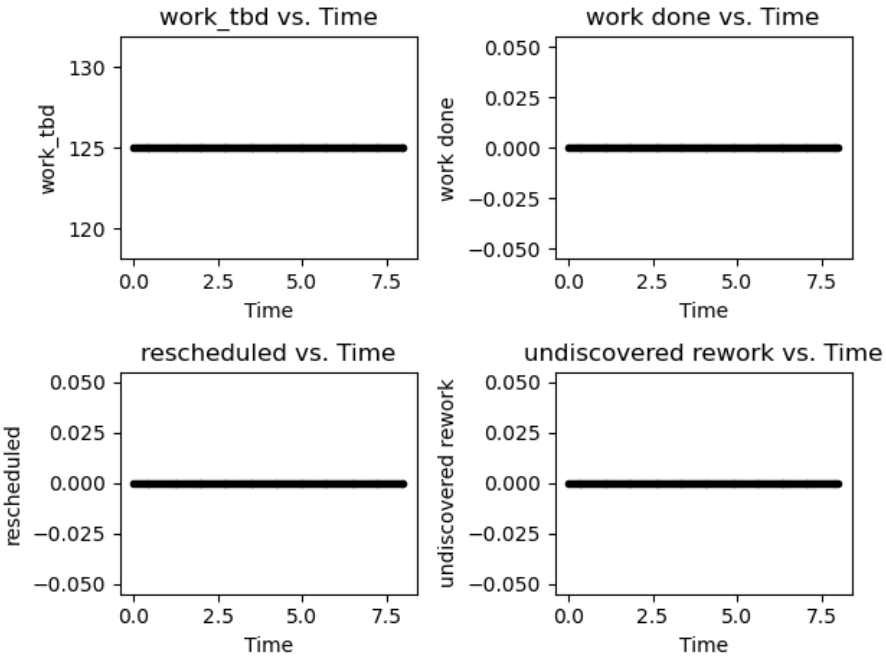


Figure B.49: Walworth Case 0048

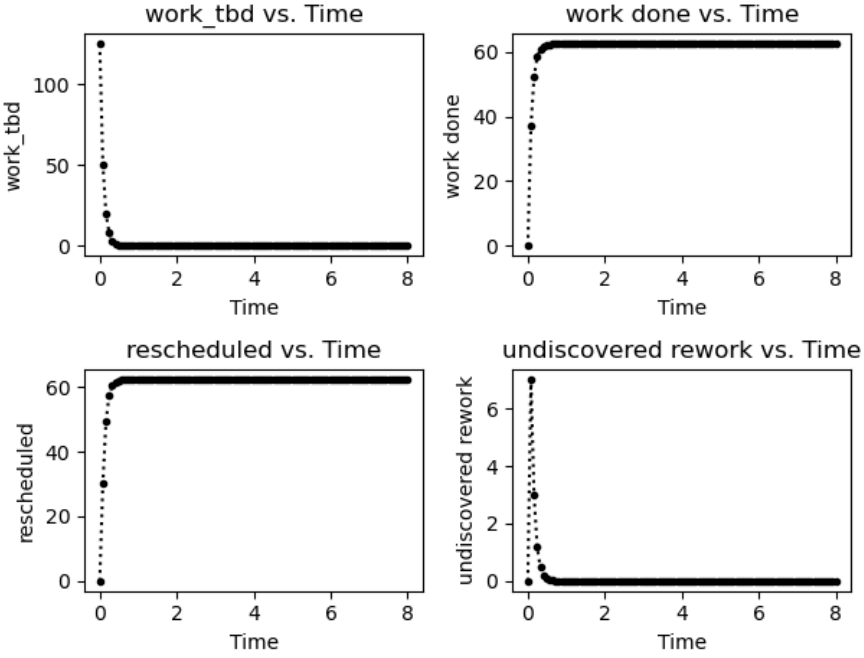


Figure B.50: Walworth Case 0049

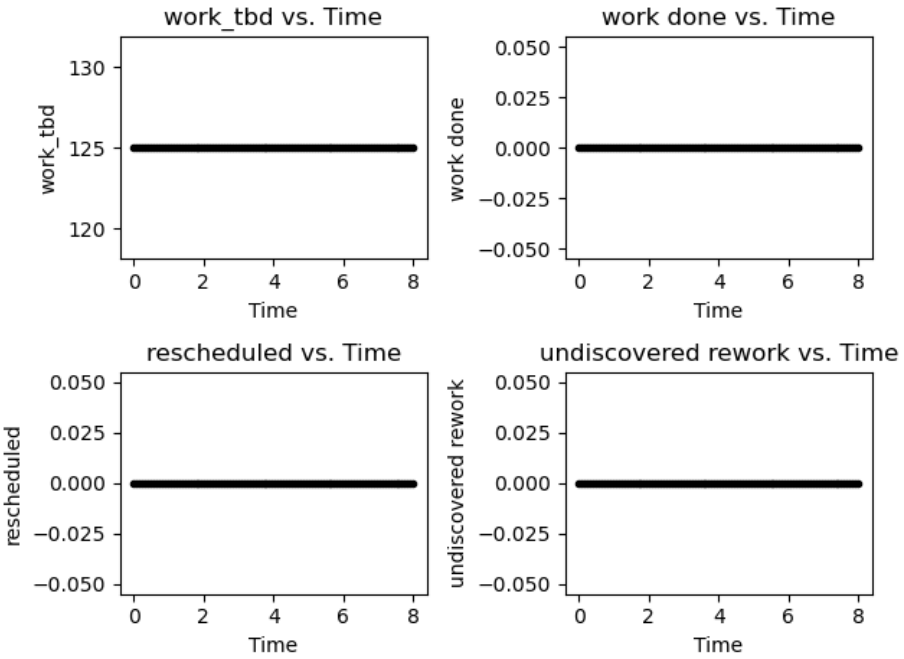


Figure B.51: Walworth Case 0050

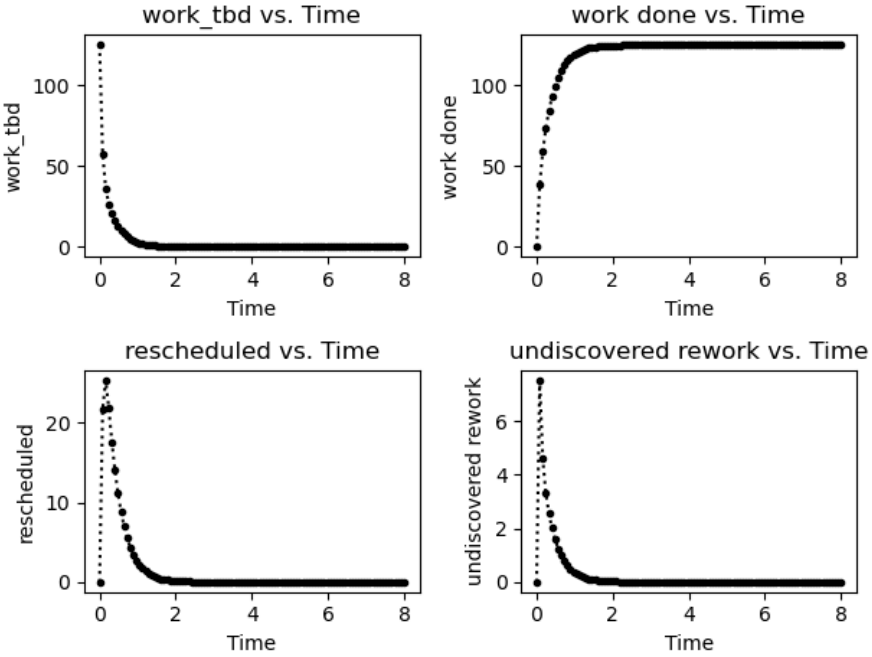


Figure B.52: Walworth Case 0051

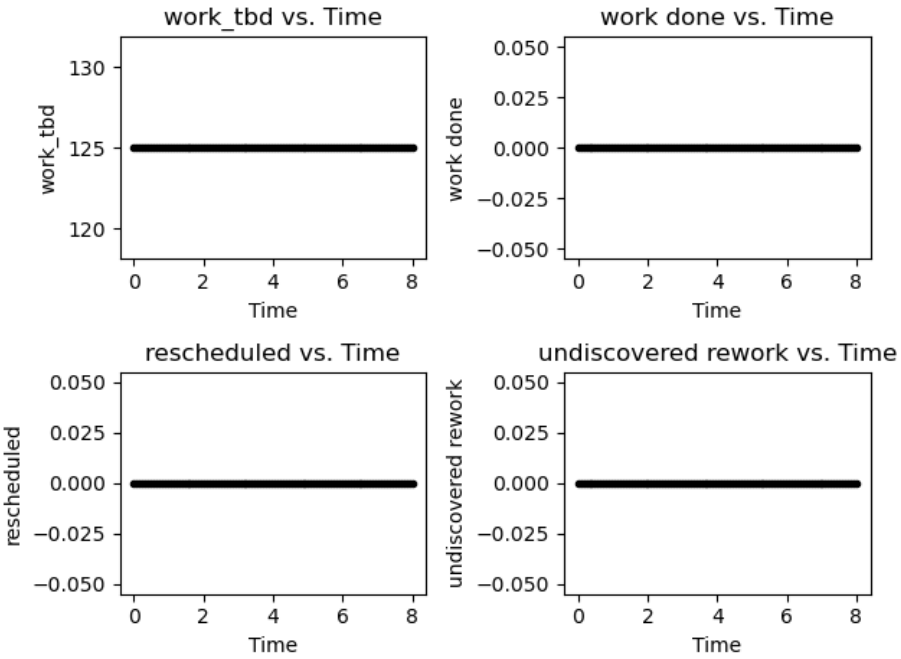


Figure B.53: Walworth Case 0052

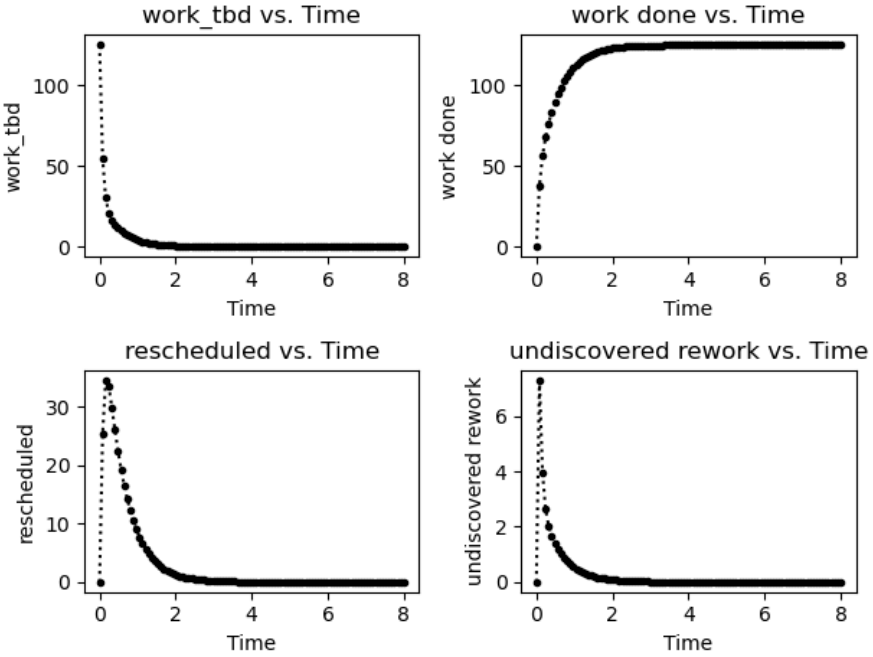


Figure B.54: Walworth Case 0053

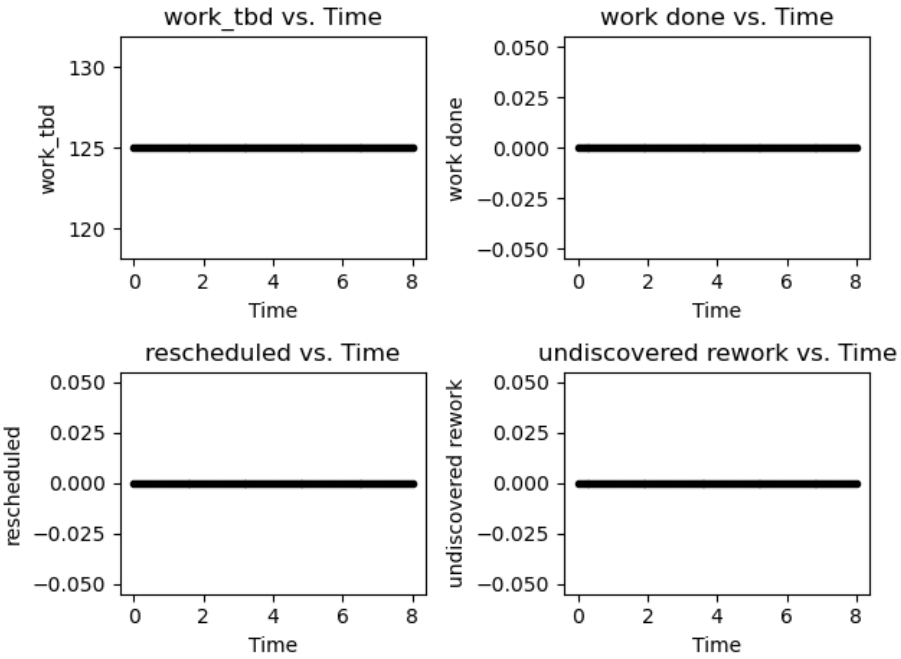


Figure B.55: Walworth Case 0054

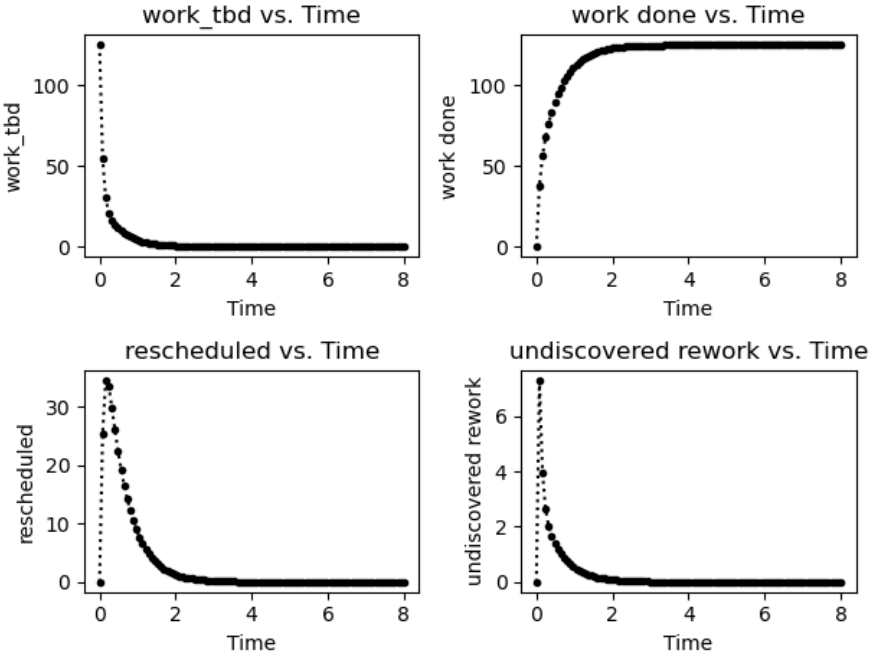


Figure B.56: Walworth Case 0055

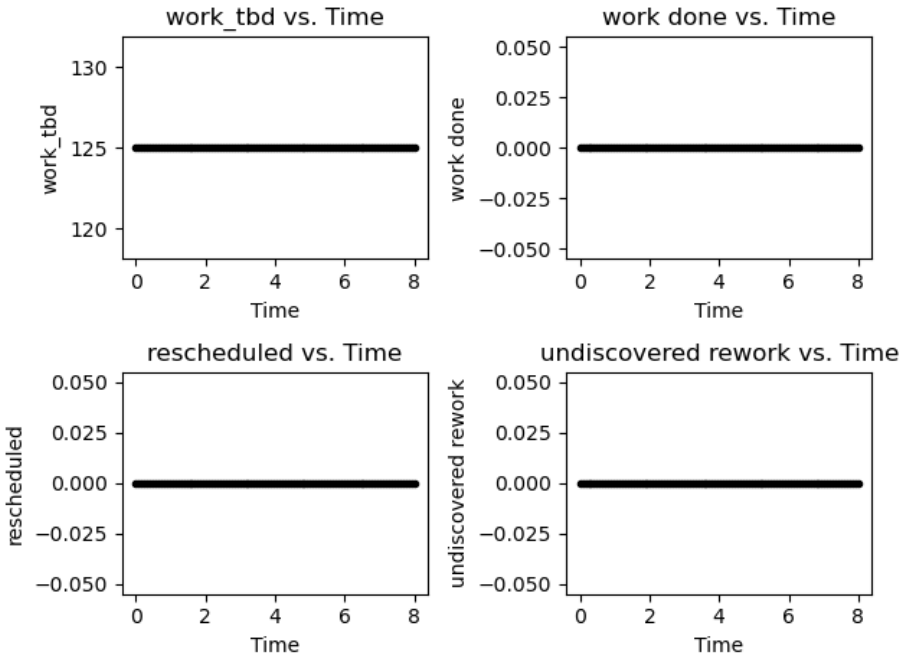


Figure B.57: Walworth Case 0056

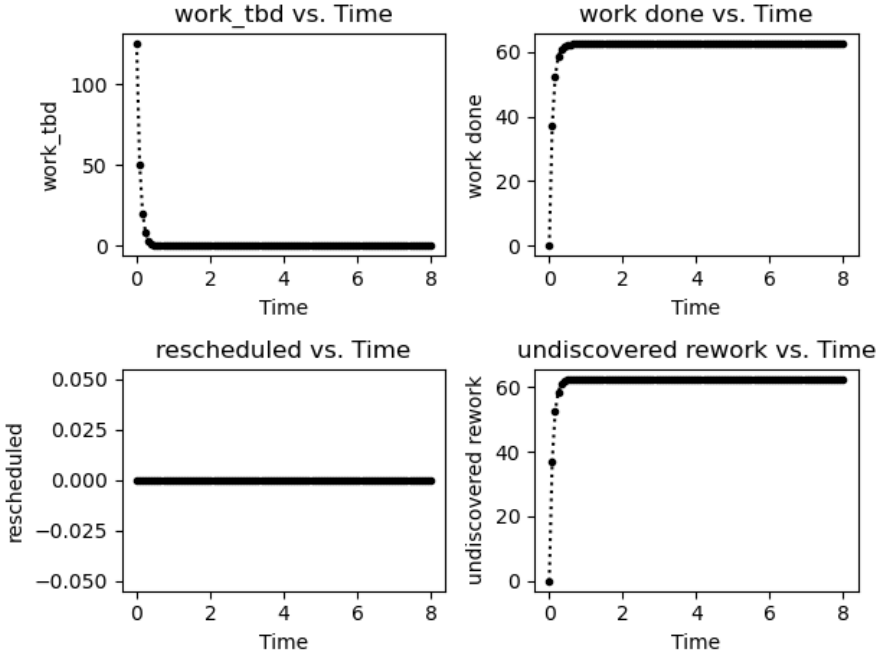


Figure B.58: Walworth Case 0057

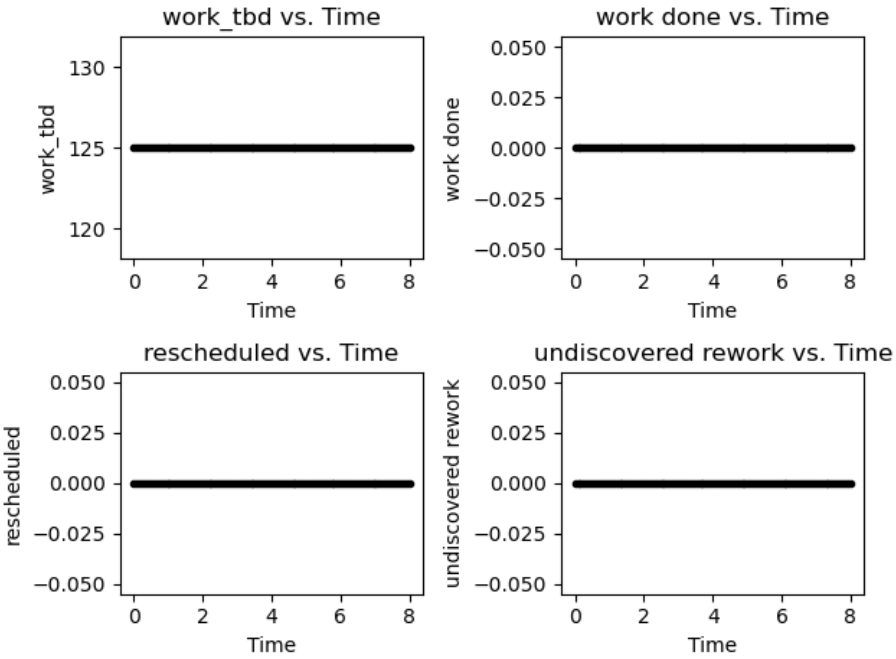


Figure B.59: Walworth Case 0058

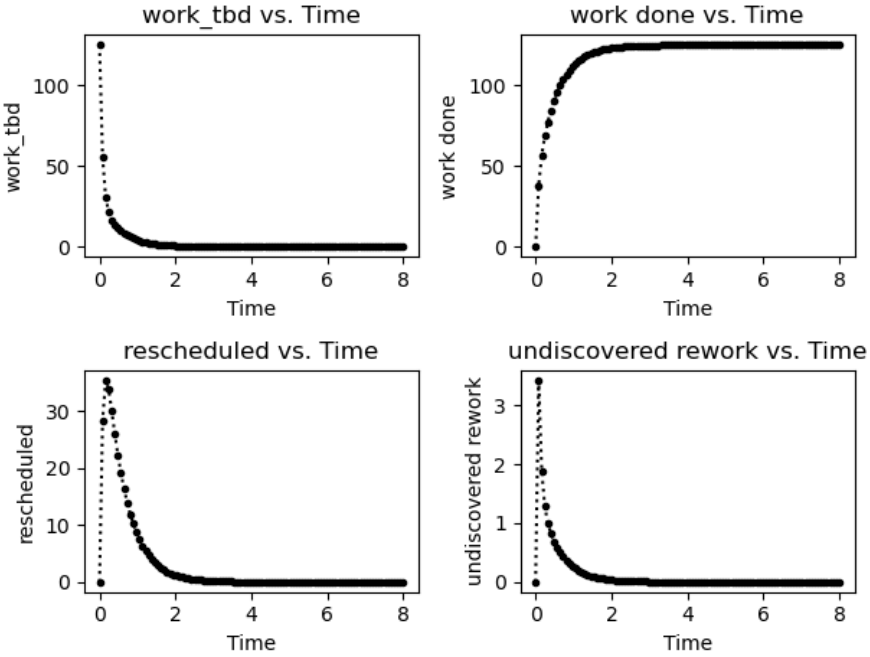


Figure B.60: Walworth Case 0059

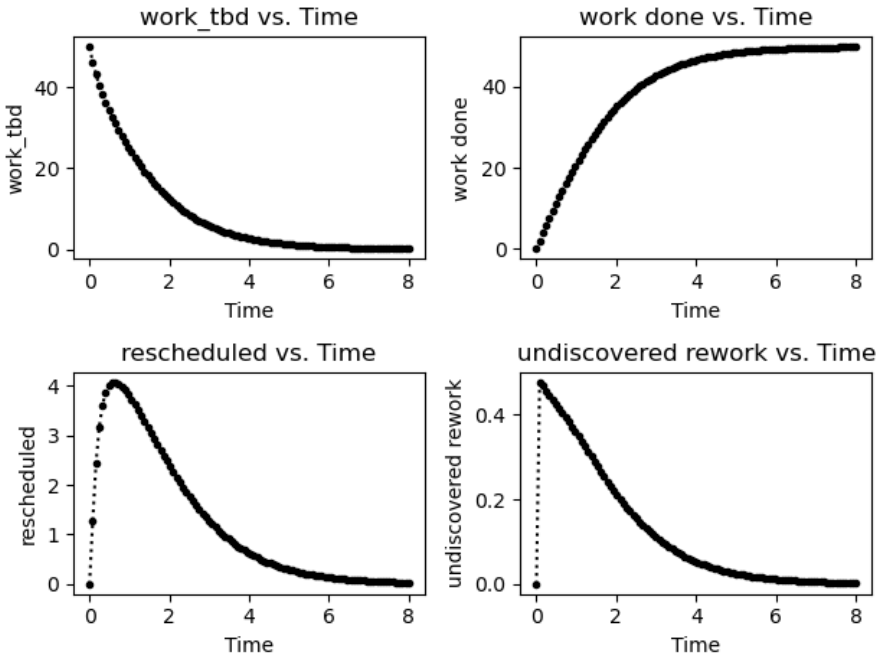


Figure B.61: Walworth Case 0060

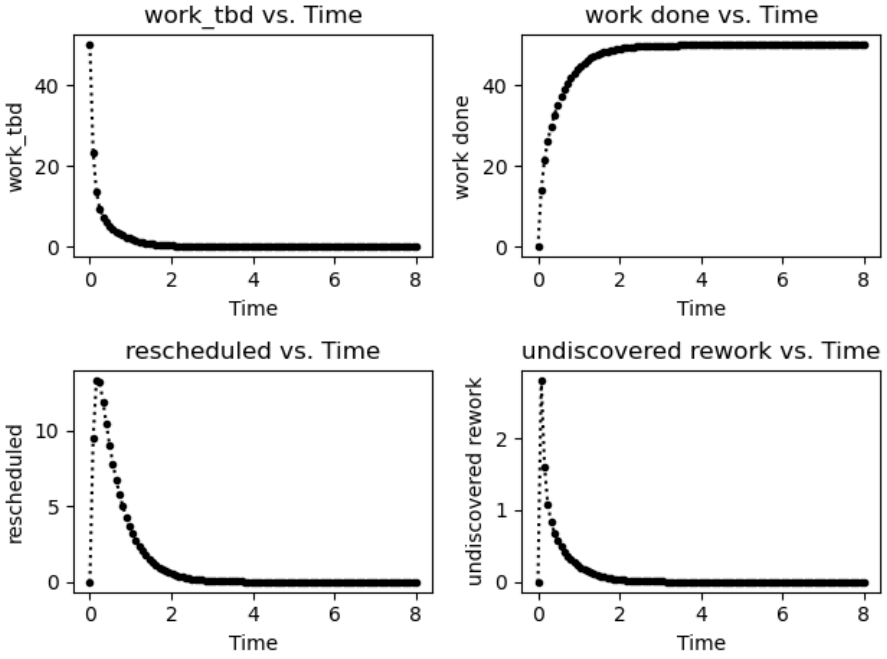


Figure B.62: Walworth Case 0061

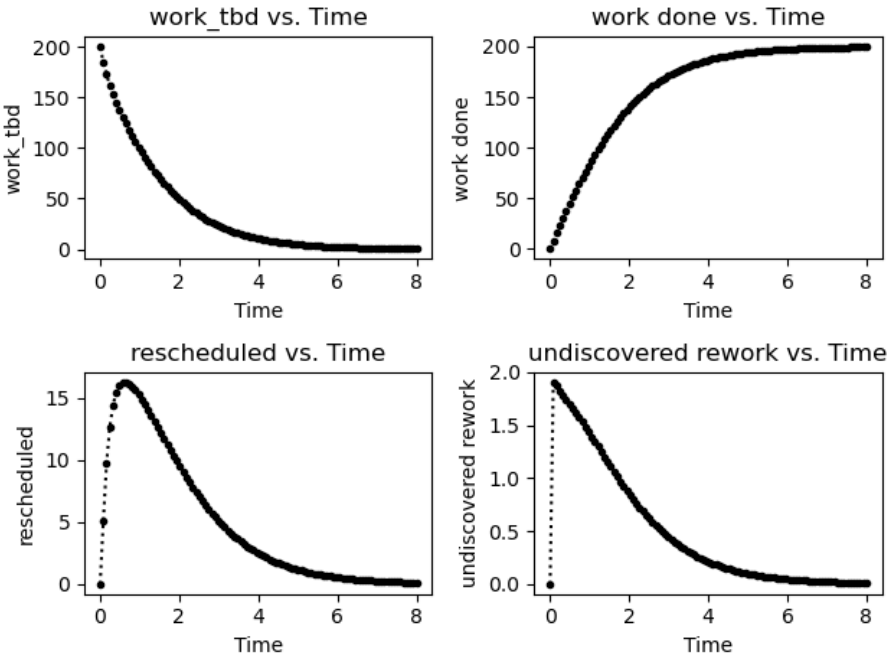


Figure B.63: Walworth Case 0062

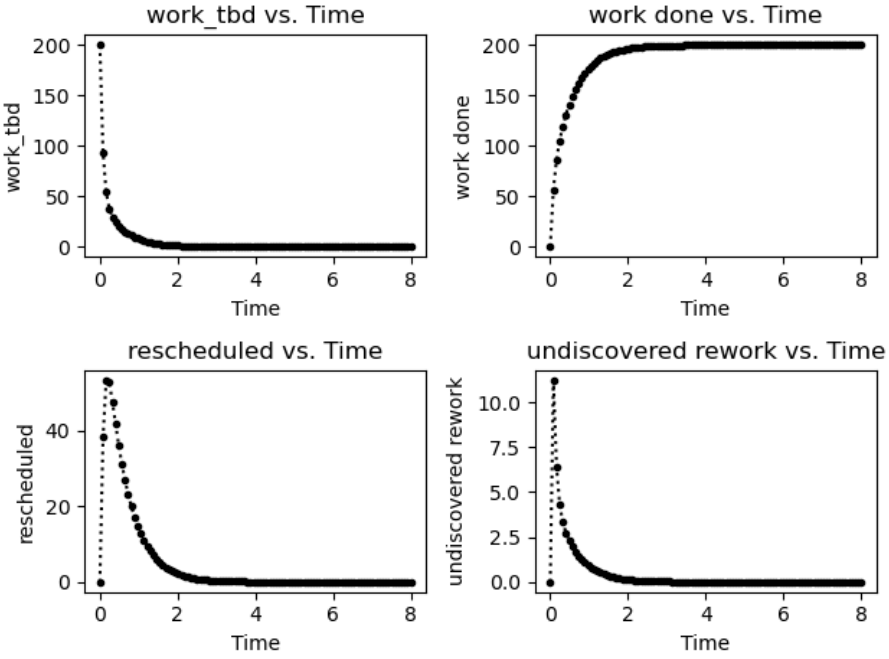


Figure B.64: Walworth Case 0063

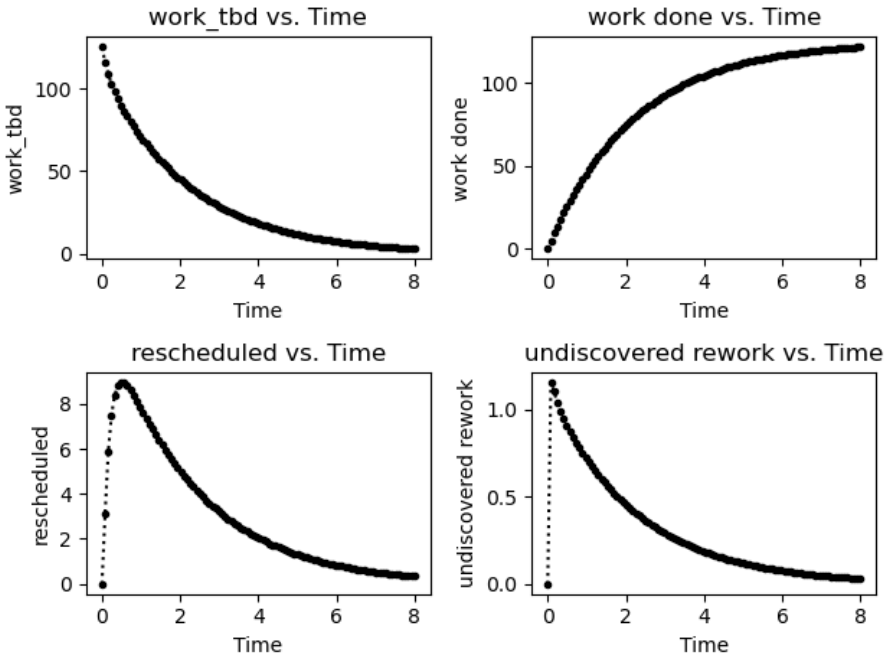


Figure B.65: Walworth Case 0064

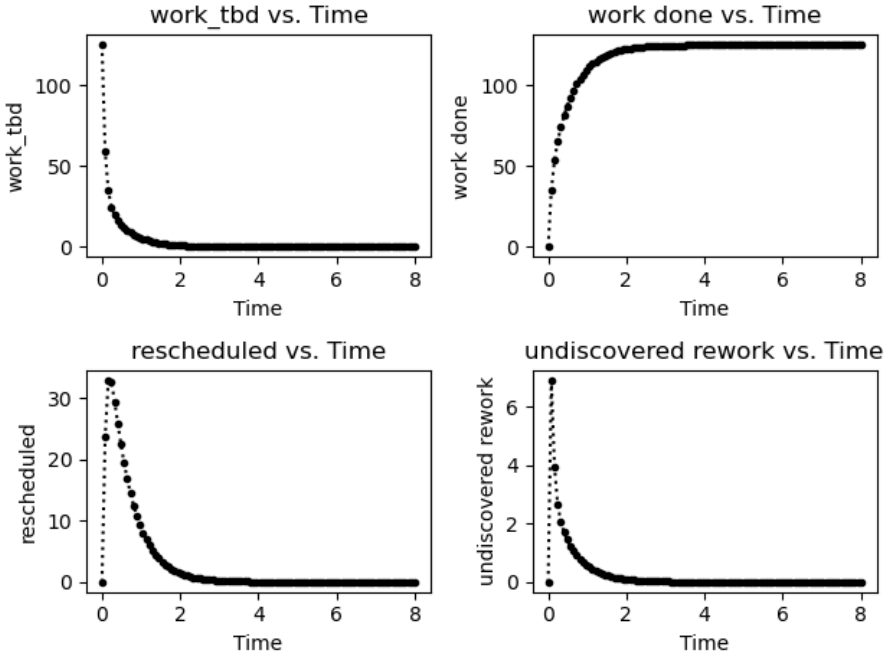


Figure B.66: Walworth Case 0065

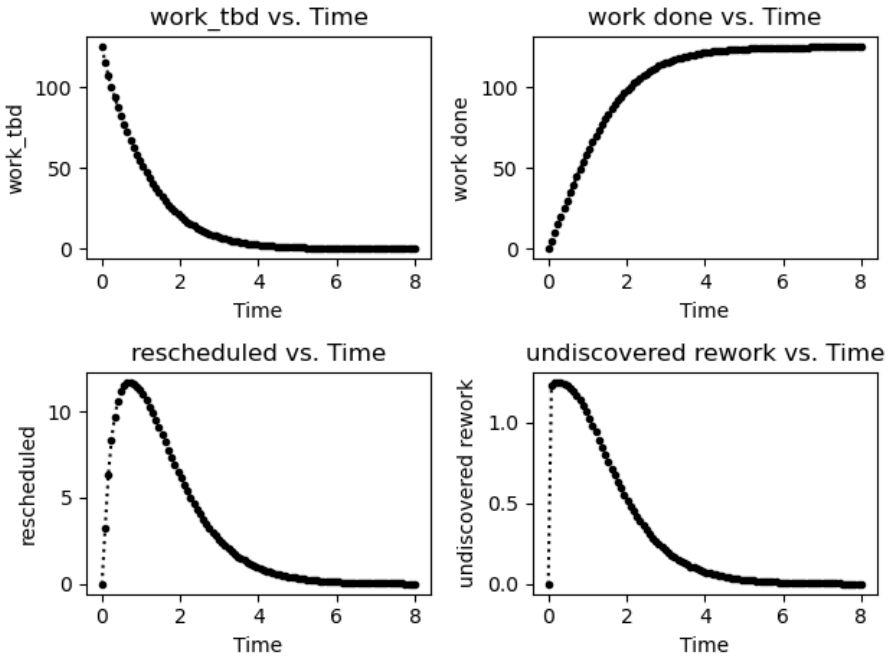


Figure B.67: Walworth Case 0066

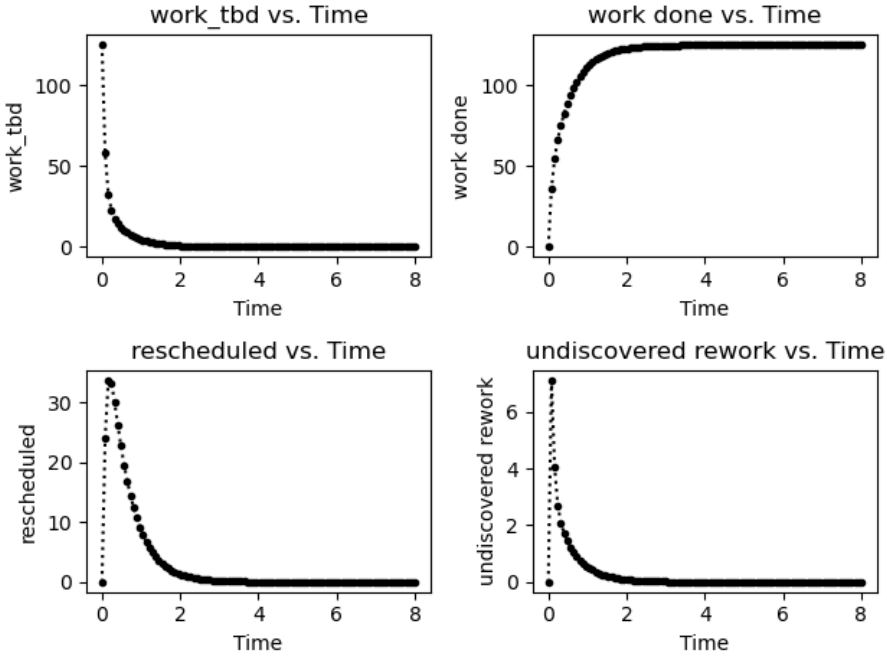


Figure B.68: Walworth Case 0067

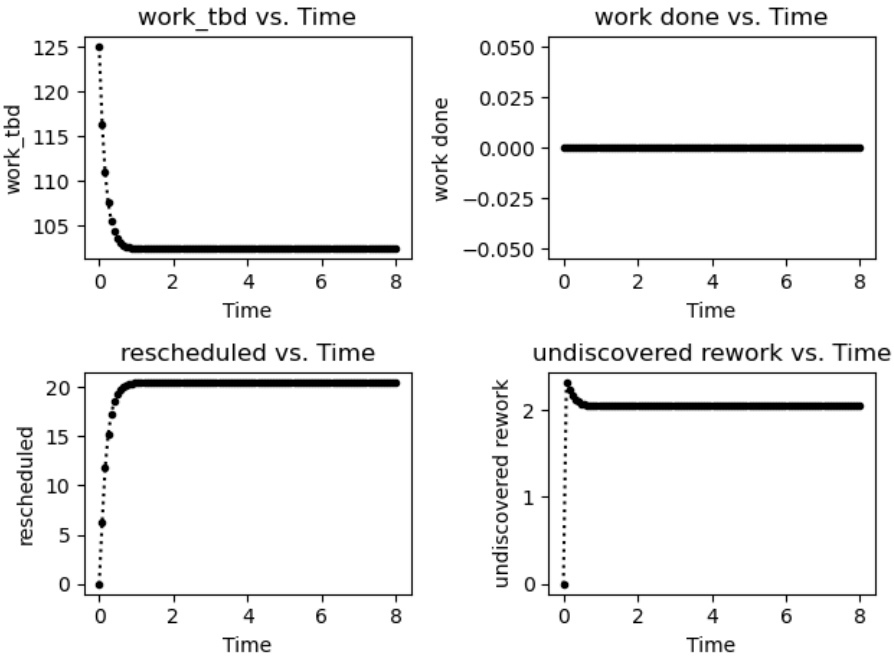


Figure B.69: Walworth Case 0068

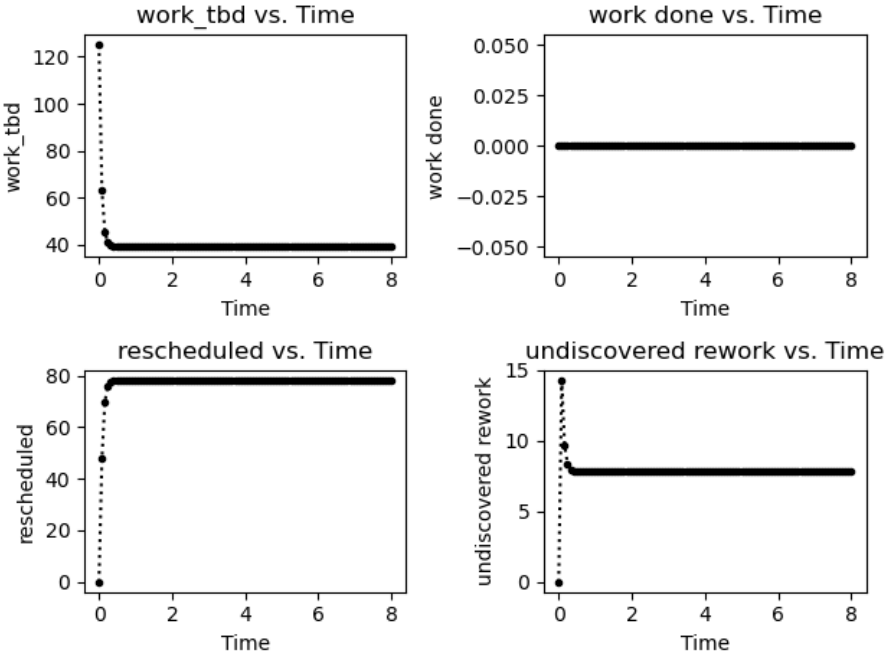


Figure B.70: Walworth Case 0069

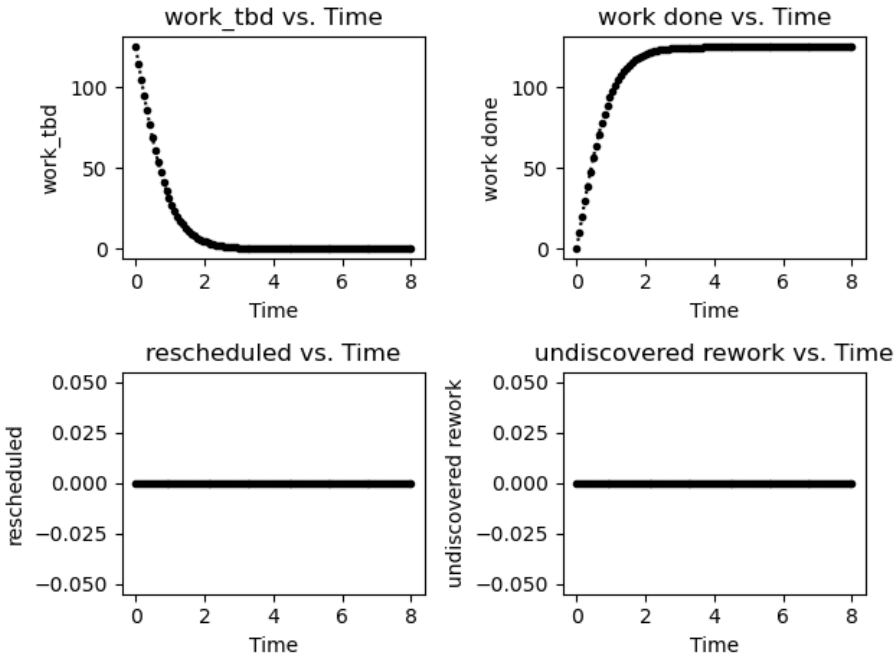


Figure B.71: Walworth Case 0070

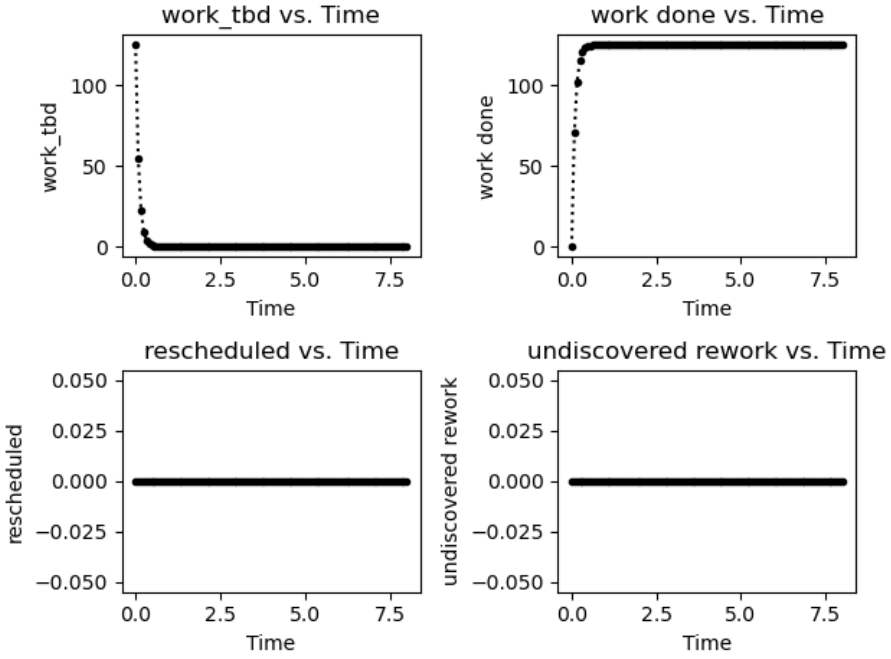


Figure B.72: Walworth Case 0071

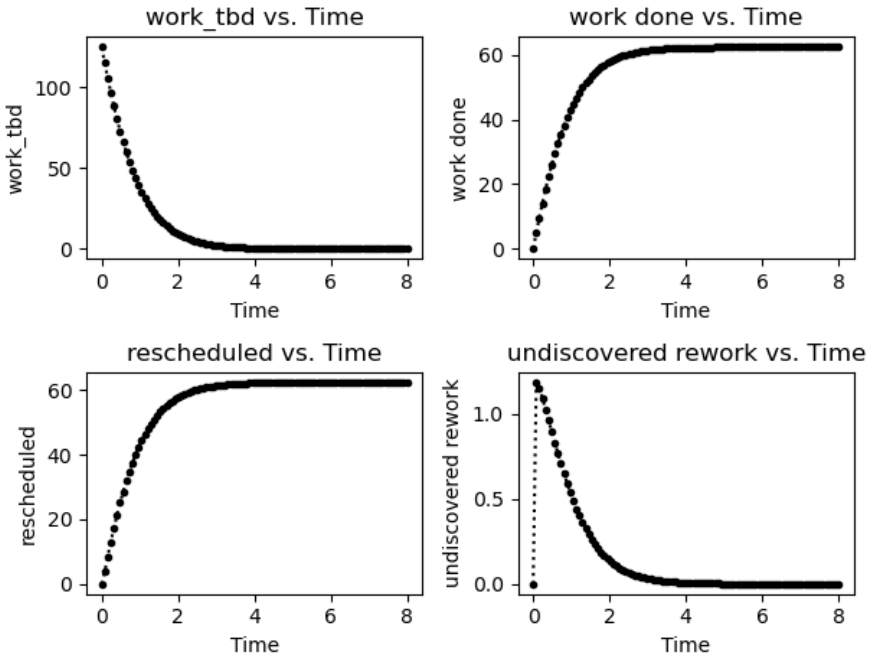


Figure B.73: Walworth Case 0072

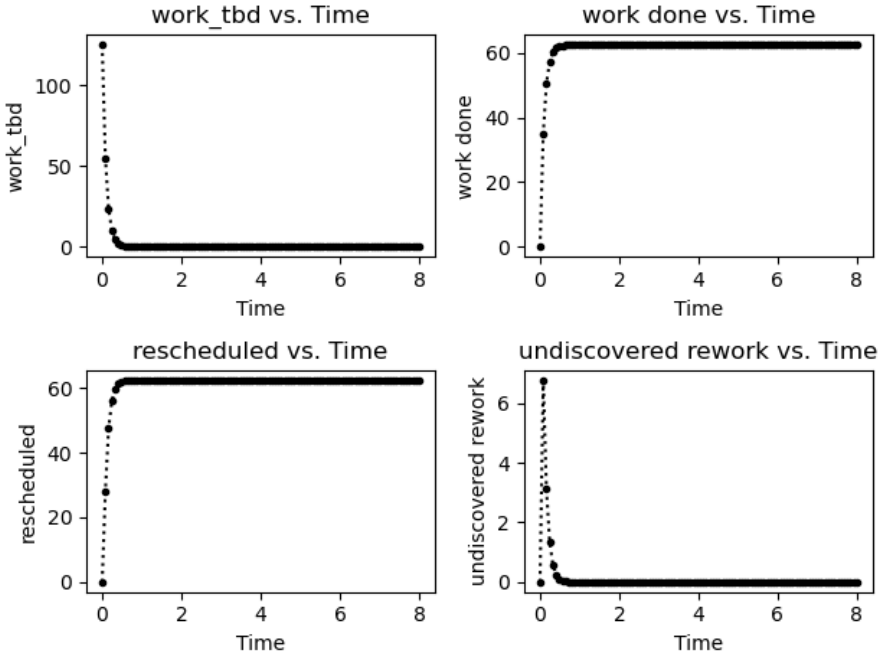


Figure B.74: Walworth Case 0073

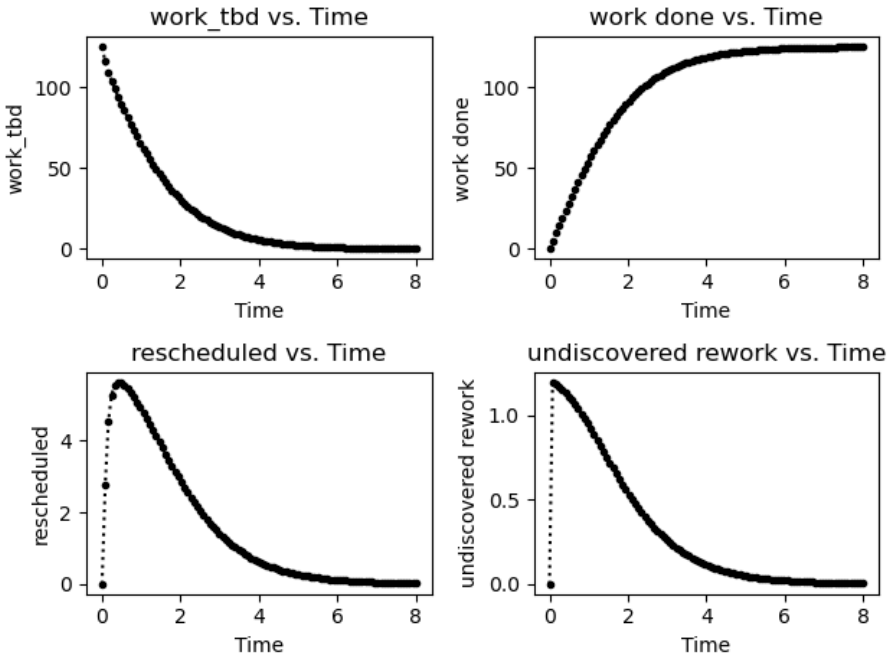


Figure B.75: Walworth Case 0074

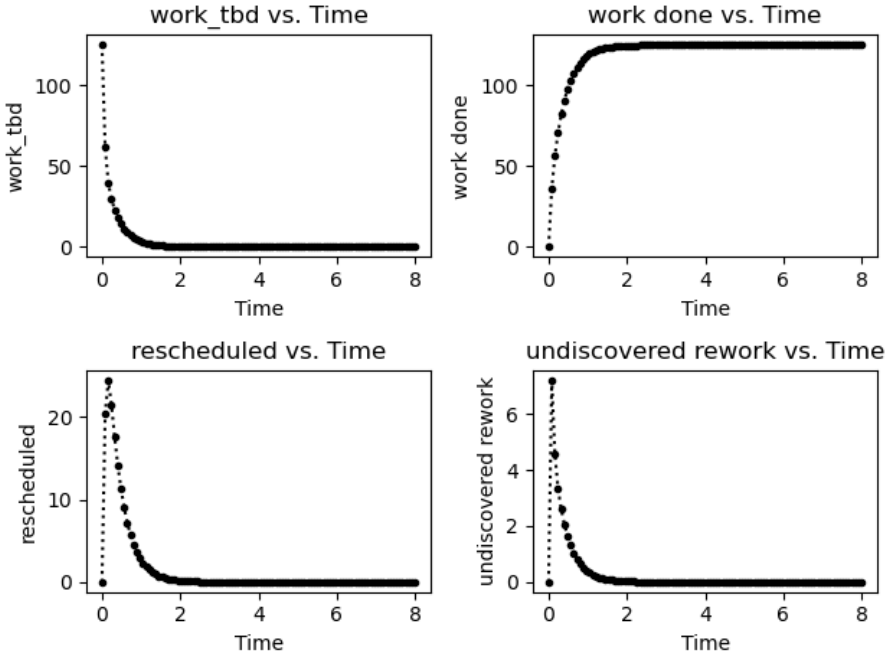


Figure B.76: Walworth Case 0075

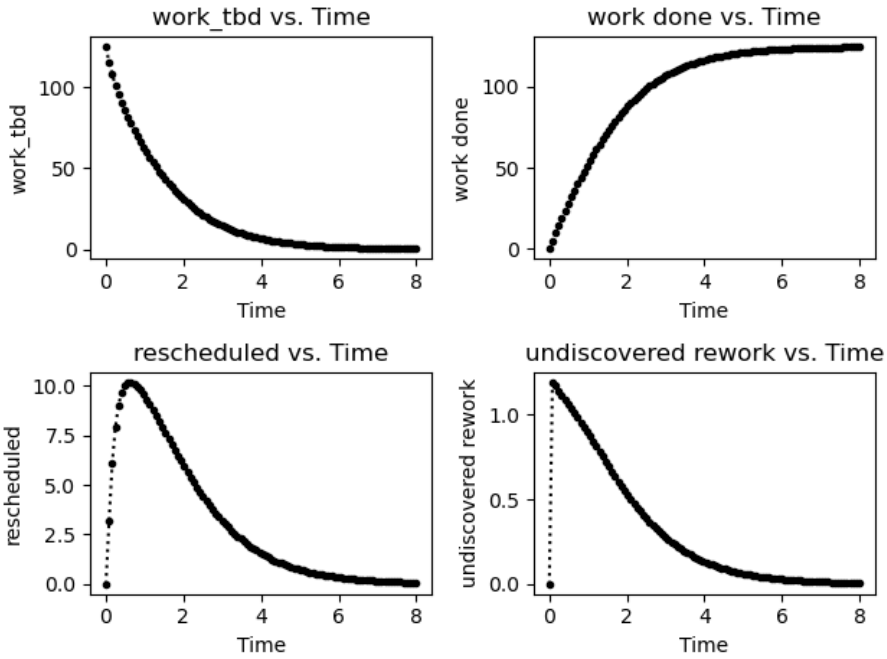


Figure B.77: Walworth Case 0076

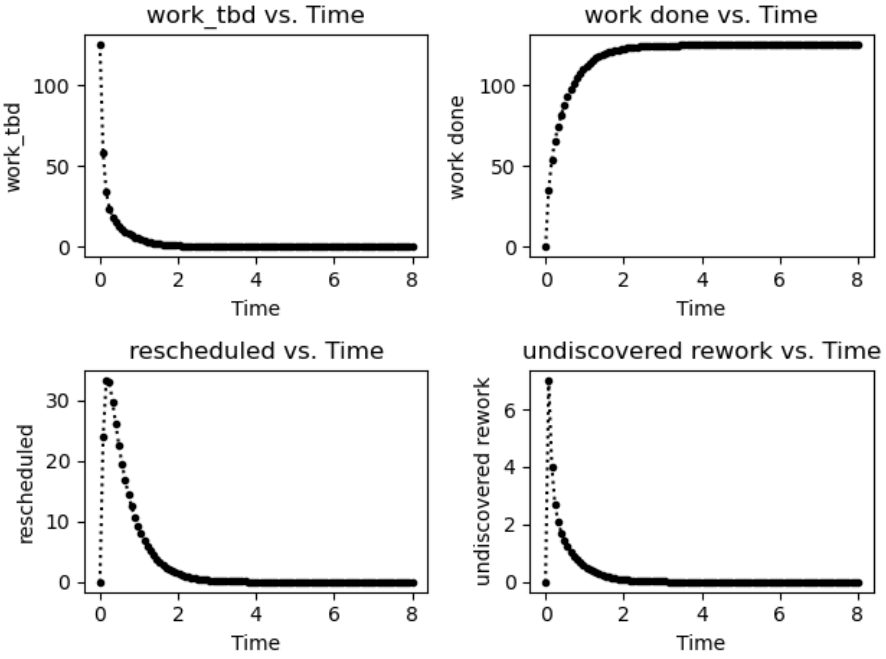


Figure B.78: Walworth Case 0077

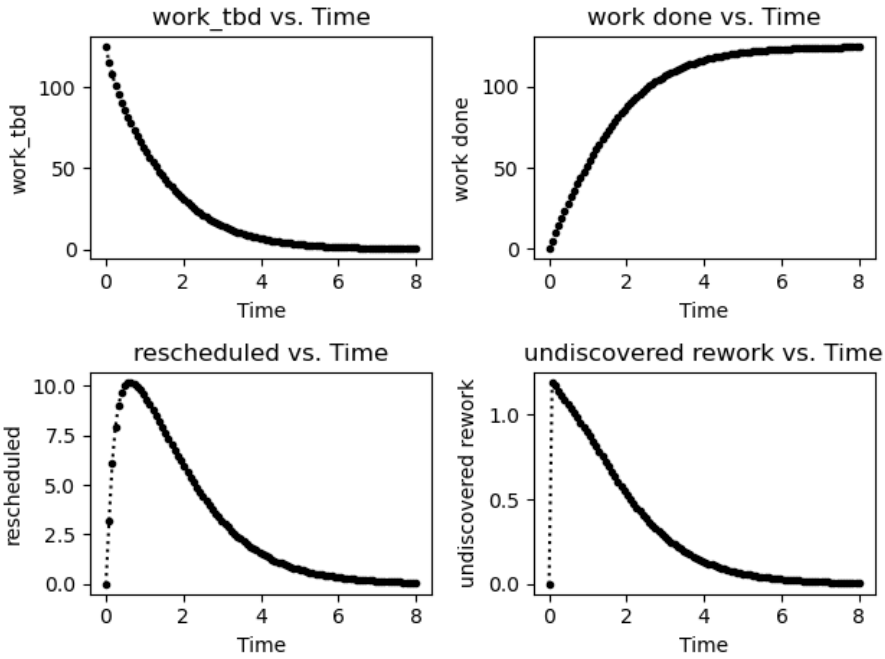


Figure B.79: Walworth Case 0078

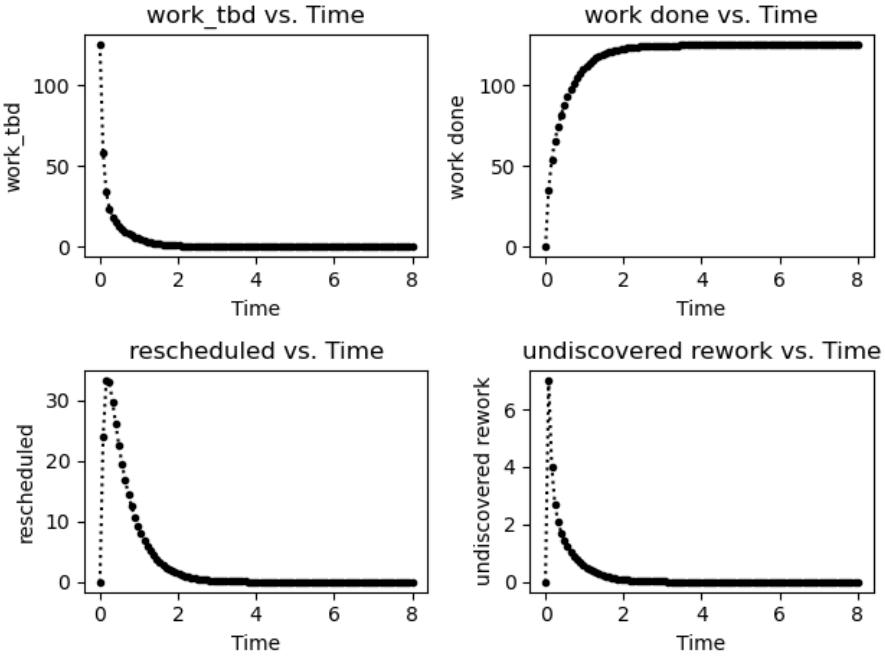


Figure B.80: Walworth Case 0079

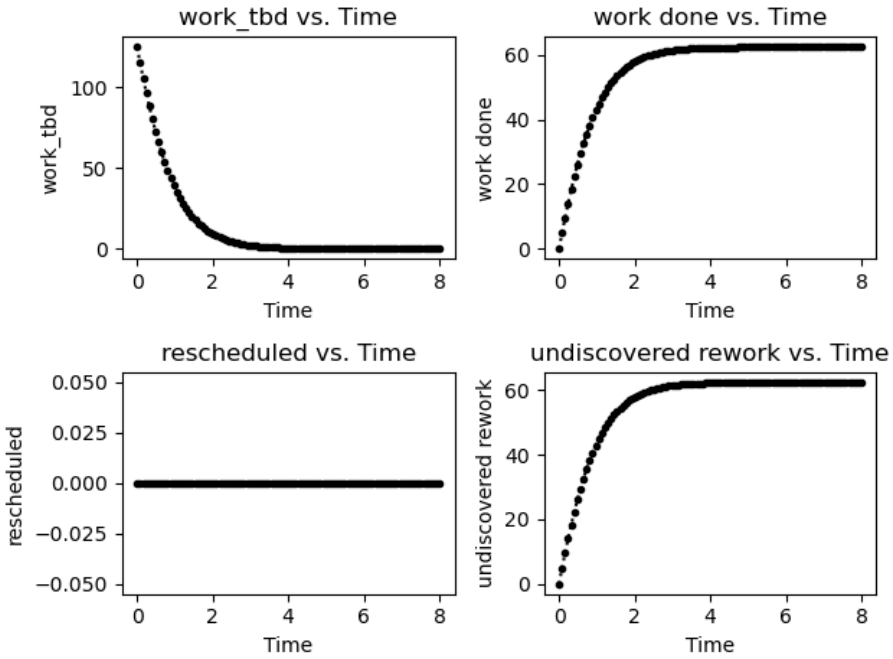


Figure B.81: Walworth Case 0080

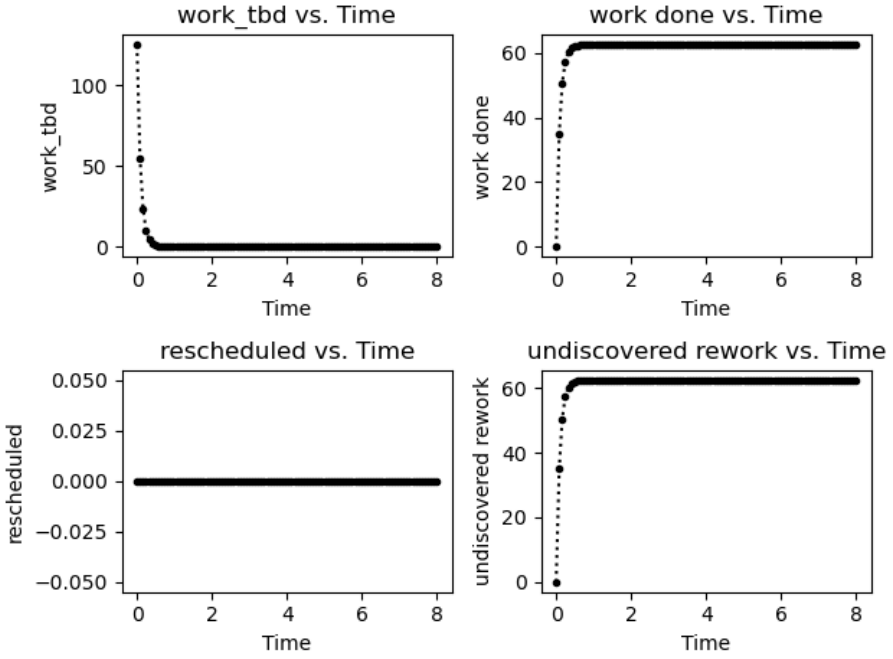


Figure B.82: Walworth Case 0081

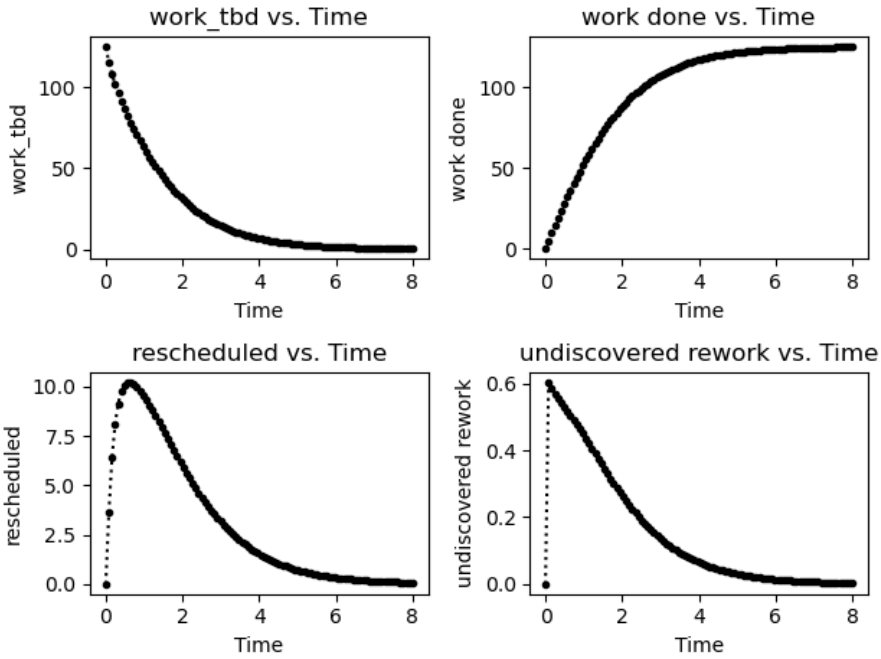


Figure B.83: Walworth Case 0082

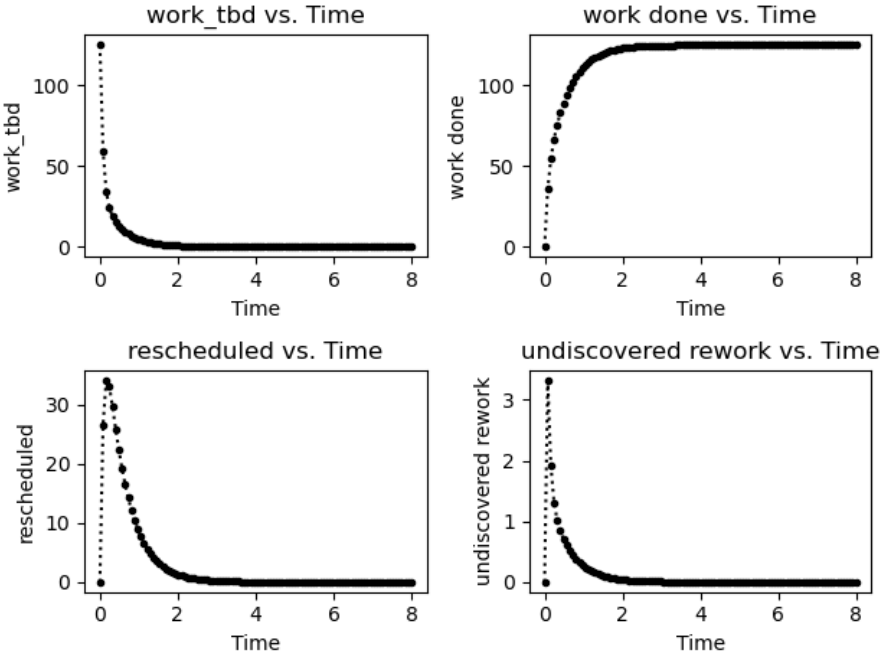


Figure B.84: Walworth Case 0083

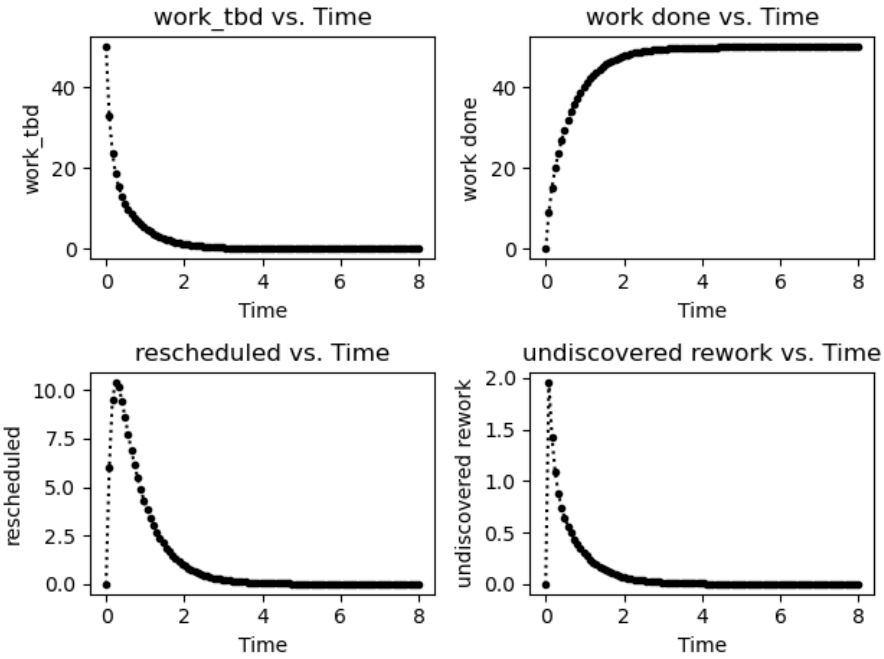


Figure B.85: Walworth Case 0084

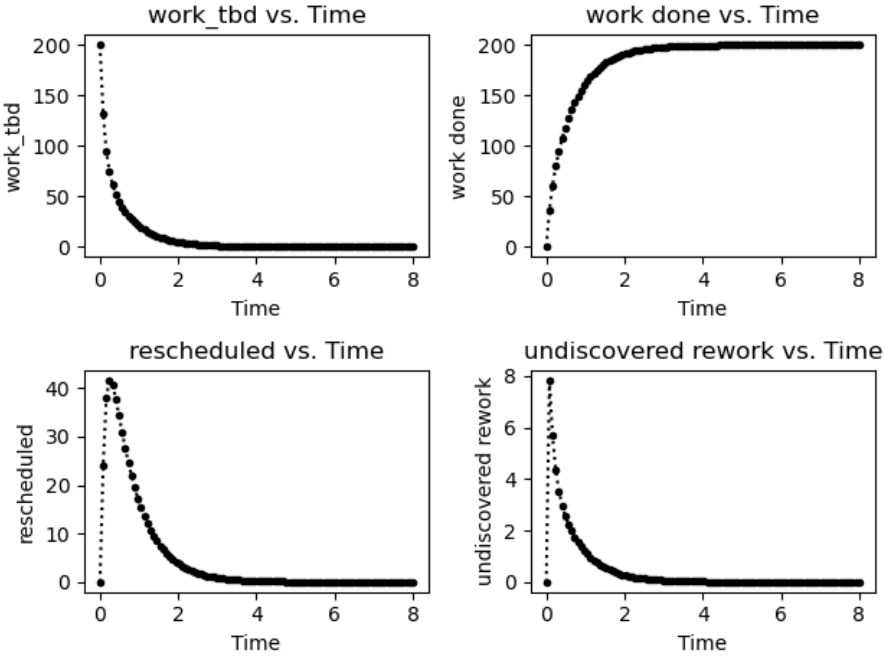


Figure B.86: Walworth Case 0085

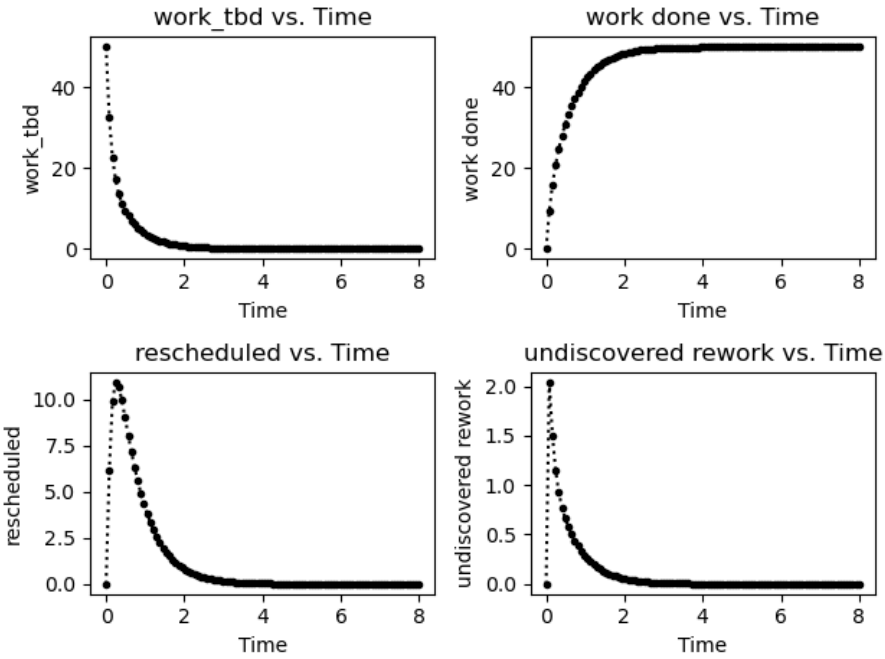


Figure B.87: Walworth Case 0086

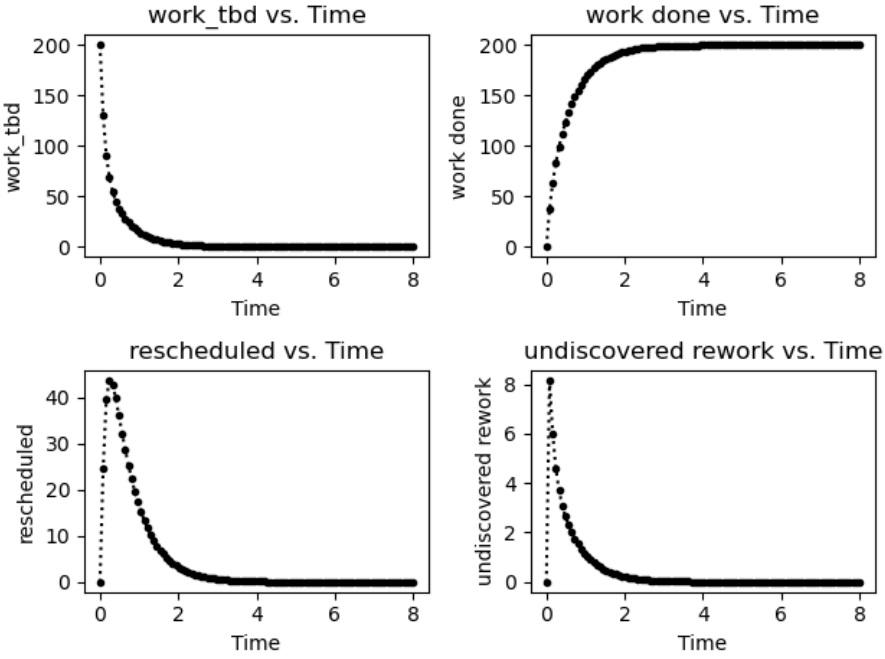


Figure B.88: Walworth Case 0087

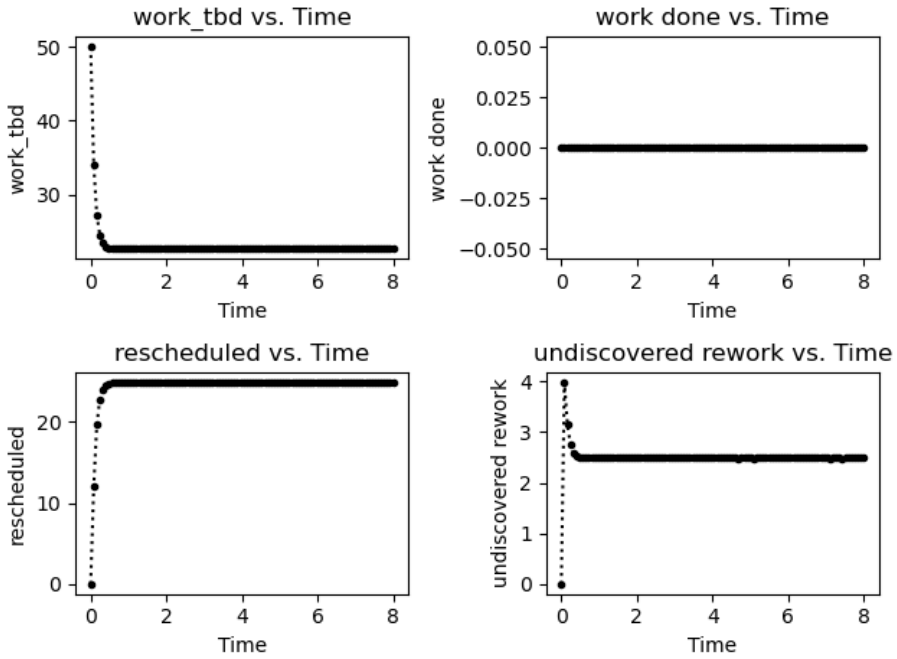


Figure B.89: Walworth Case 0088

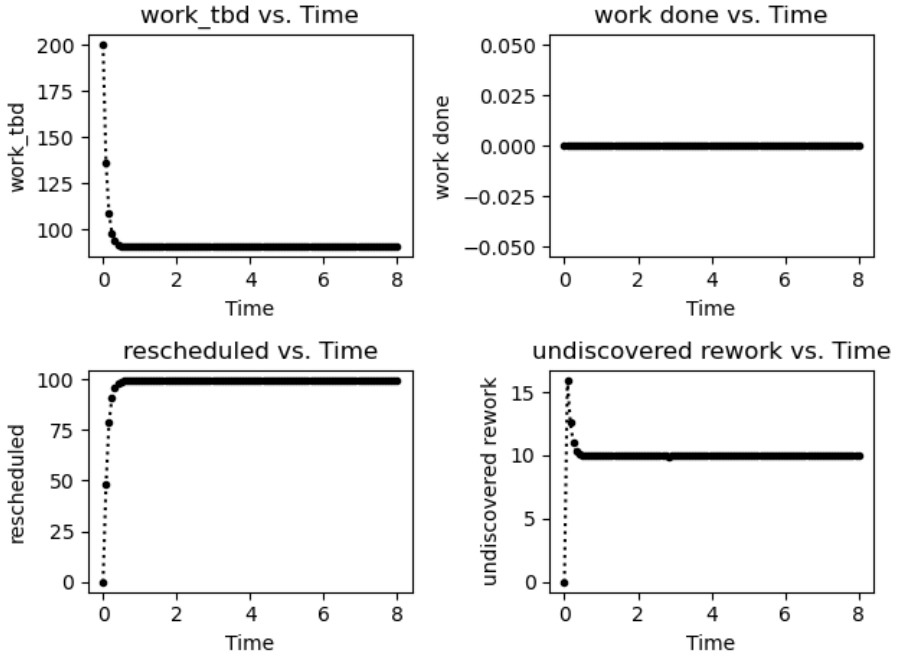


Figure B.90: Walworth Case 0089

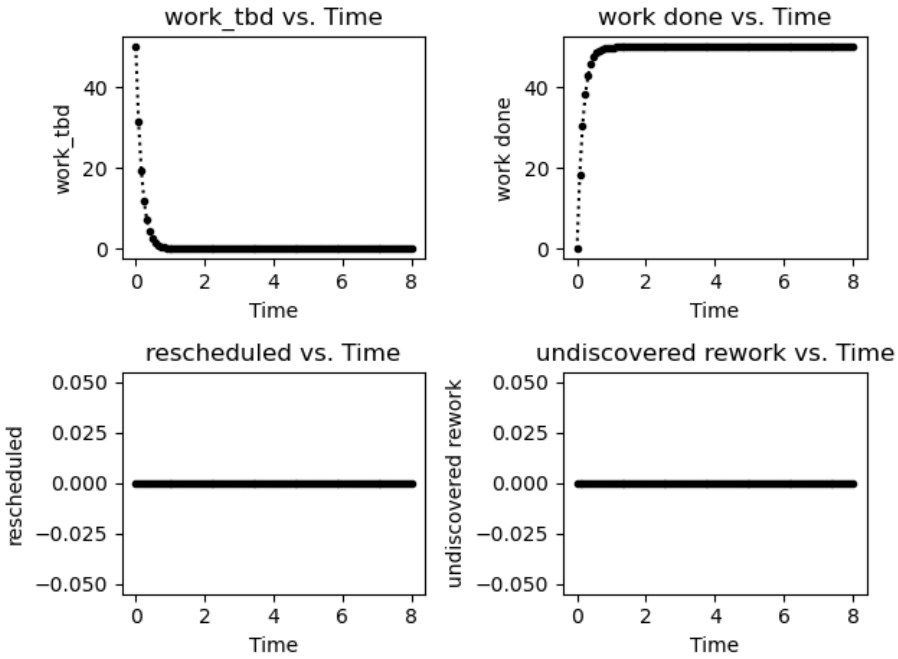


Figure B.91: Walworth Case 0090

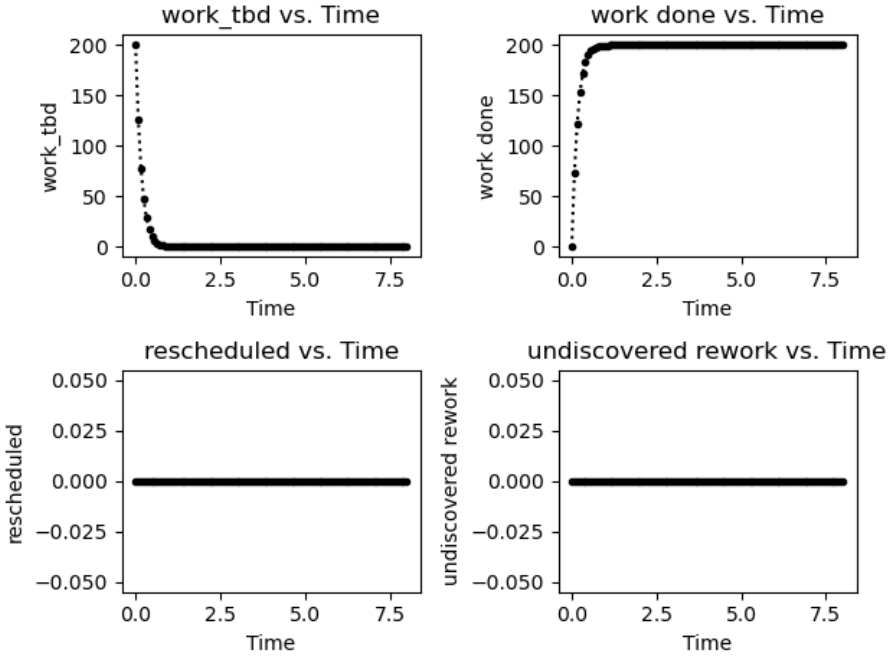


Figure B.92: Walworth Case 0091

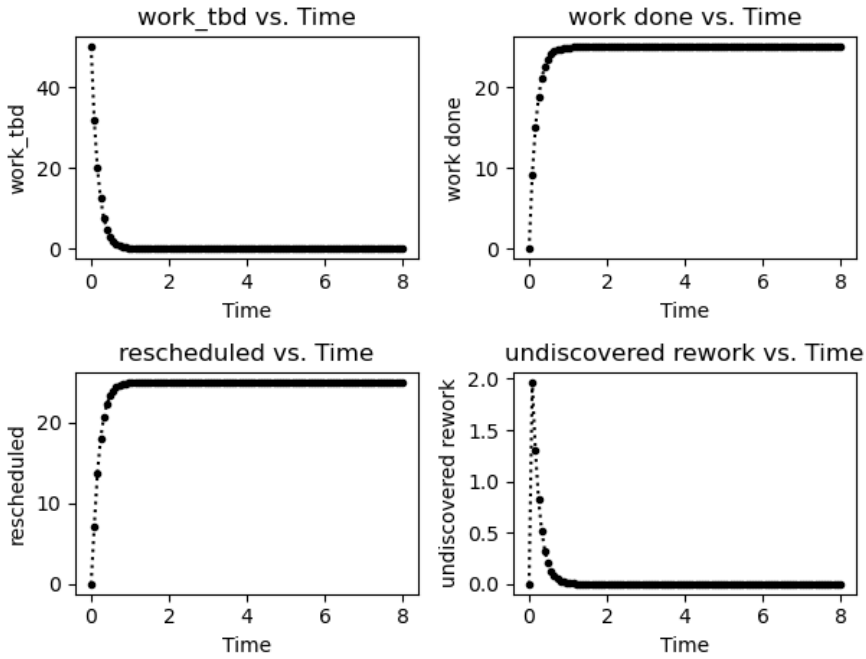


Figure B.93: Walworth Case 0092

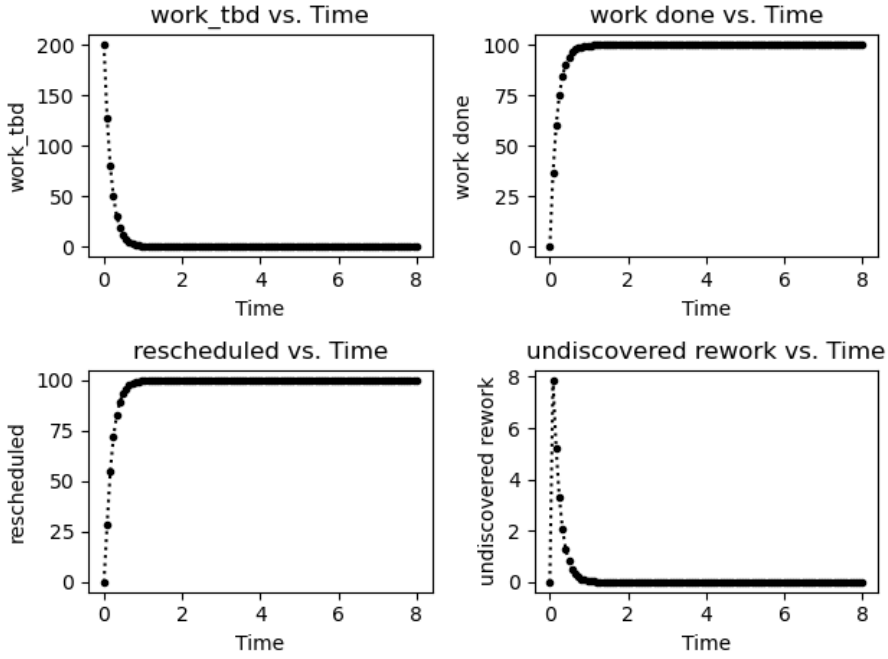


Figure B.94: Walworth Case 0093

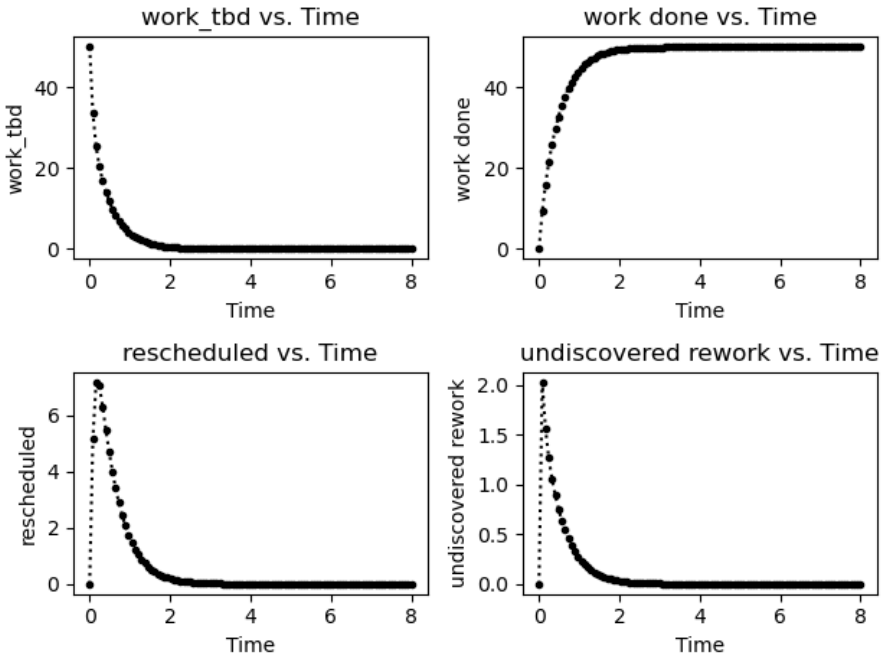


Figure B.95: Walworth Case 0094

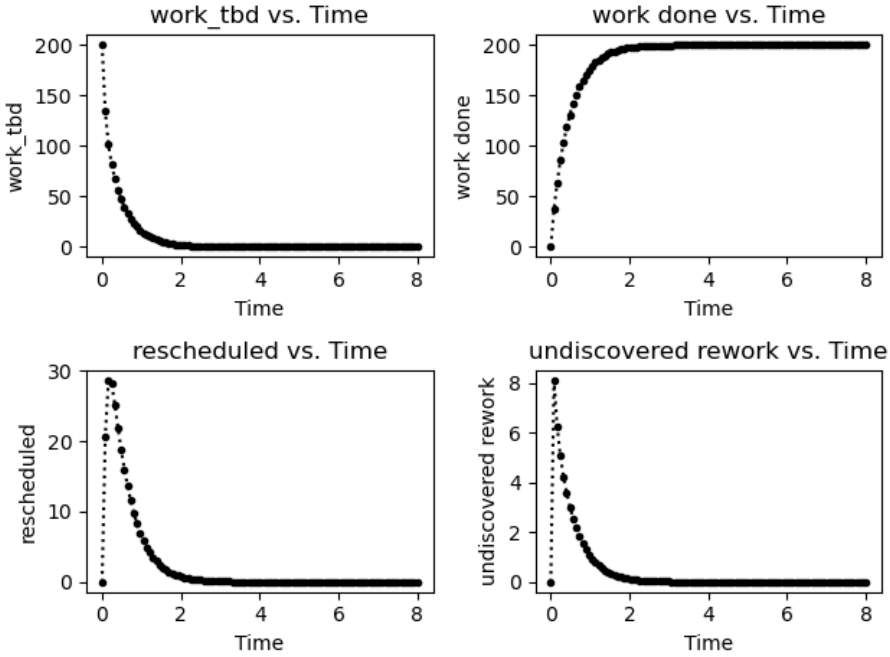


Figure B.96: Walworth Case 0095

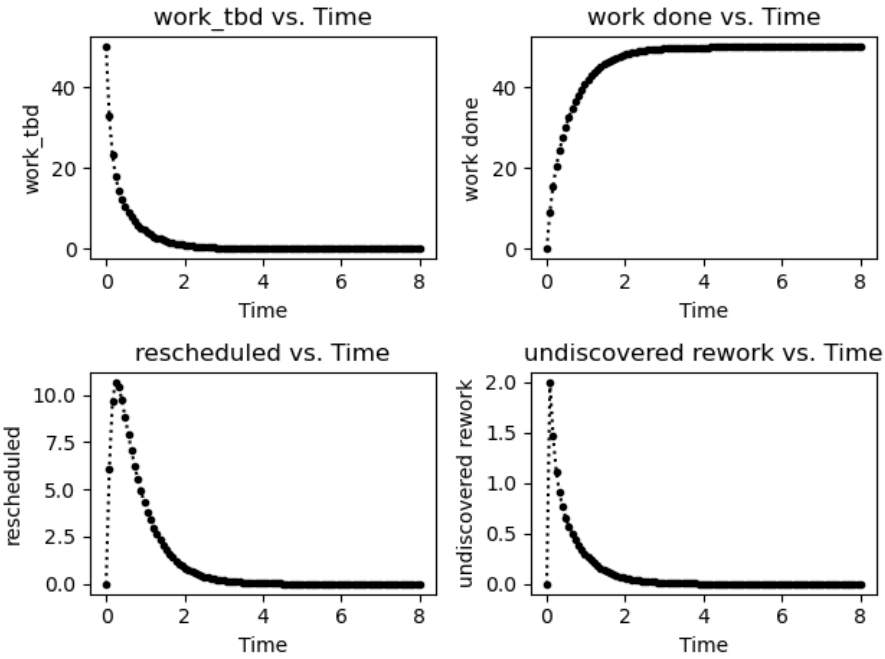


Figure B.97: Walworth Case 0096

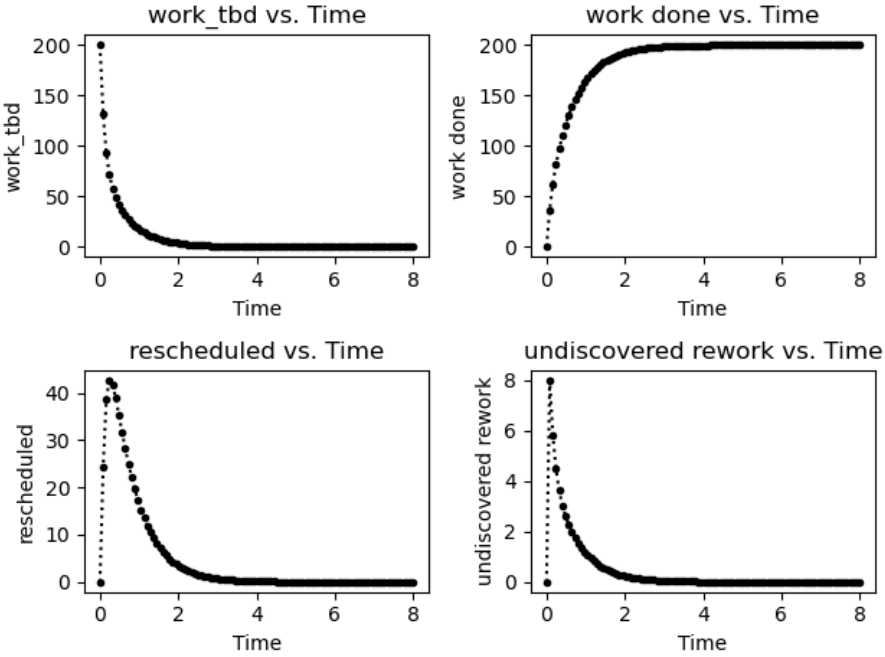


Figure B.98: Walworth Case 0097

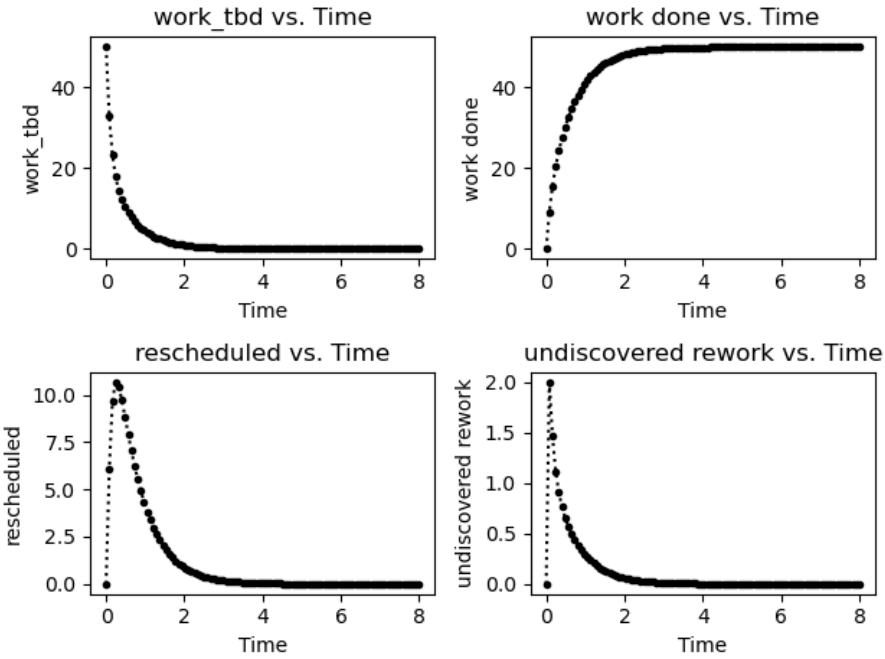


Figure B.99: Walworth Case 0098

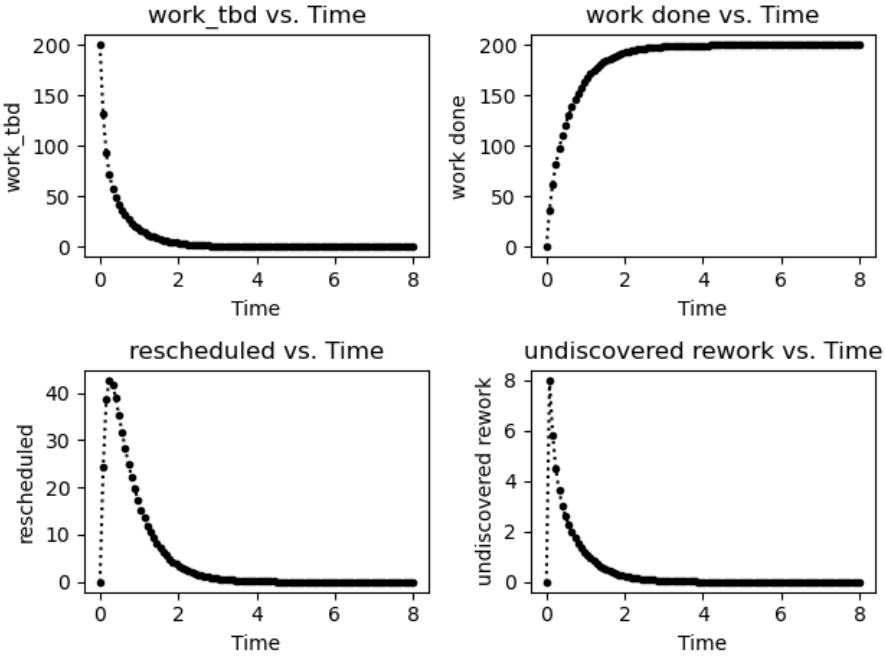


Figure B.100: Walworth Case 0099

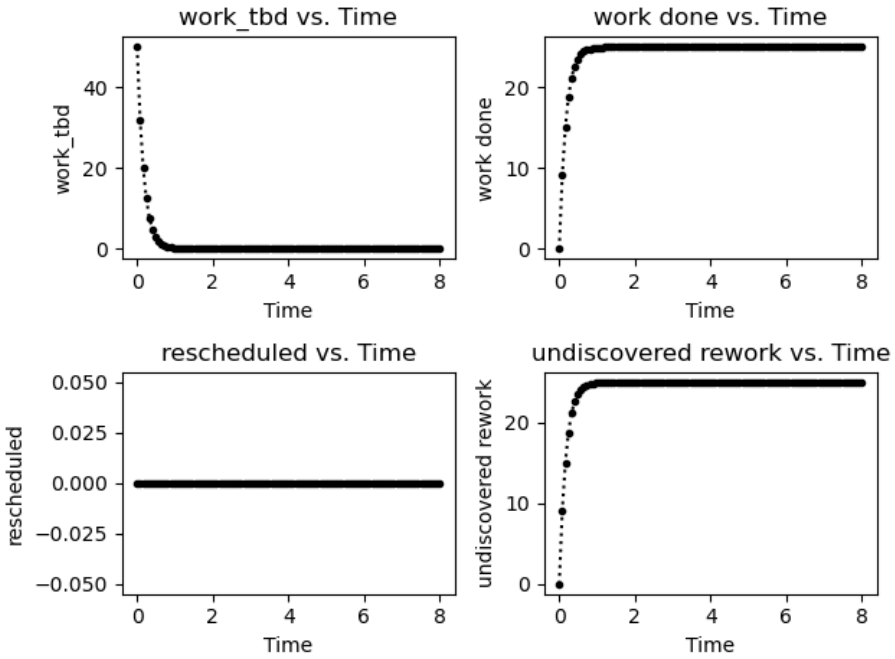


Figure B.101: Walworth Case 0100

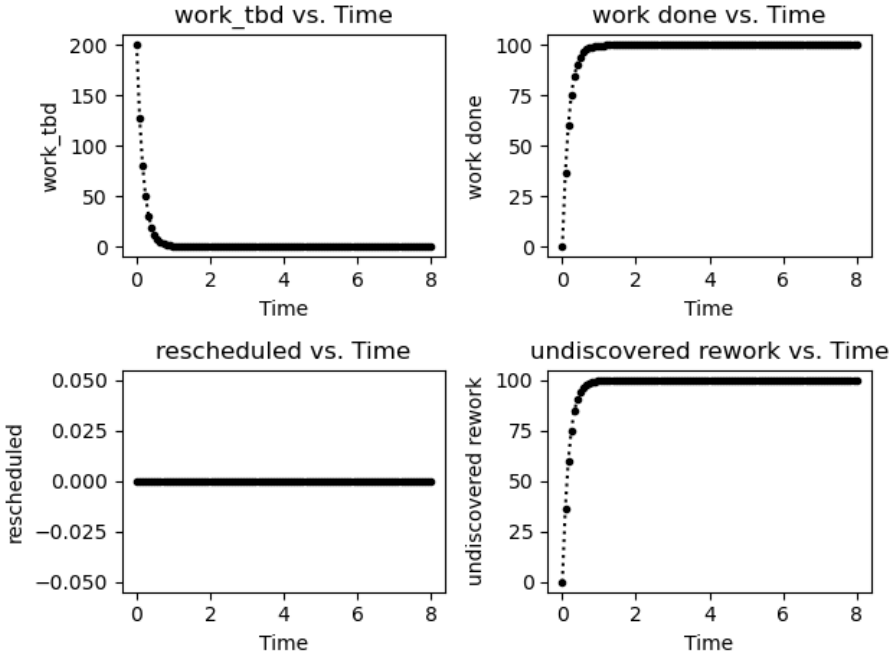


Figure B.102: Walworth Case 0101

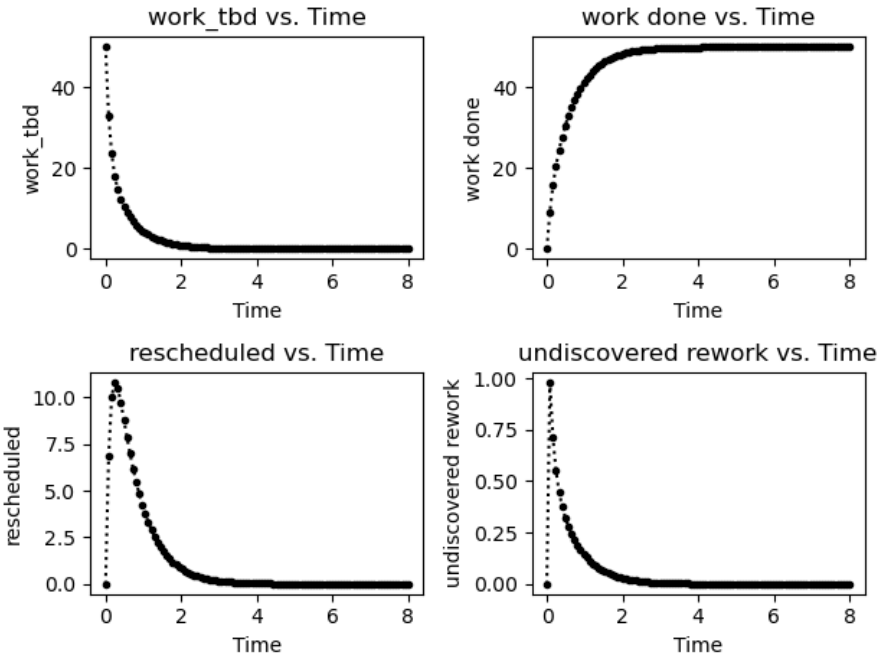


Figure B.103: Walworth Case 0102

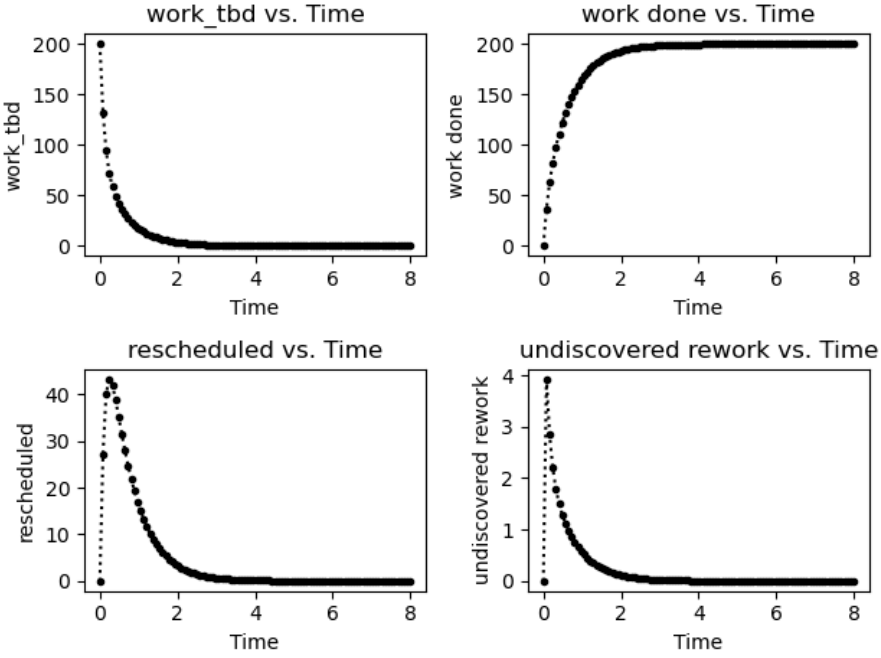


Figure B.104: Walworth Case 0103

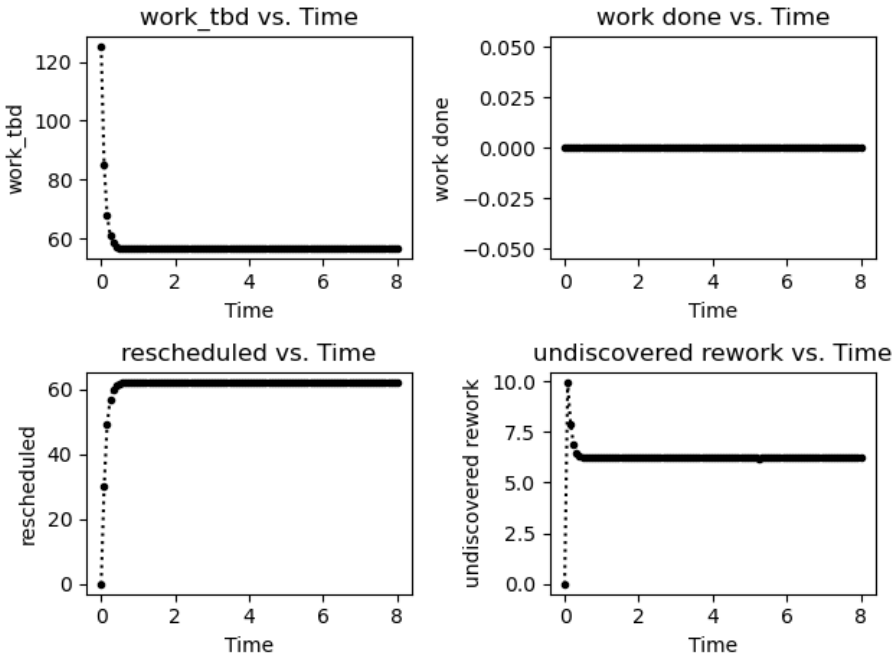


Figure B.105: Walworth Case 0104

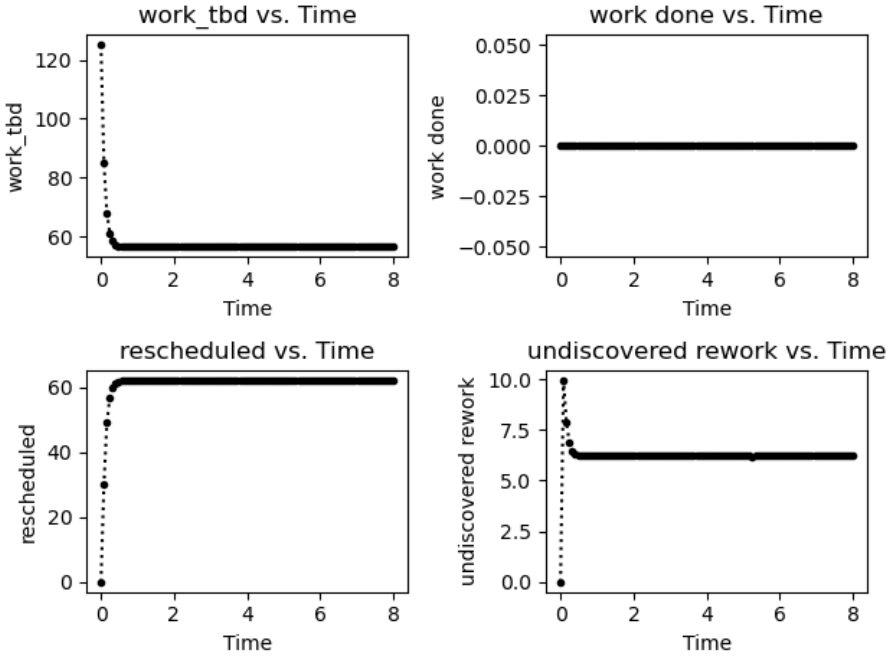


Figure B.106: Walworth Case 0105

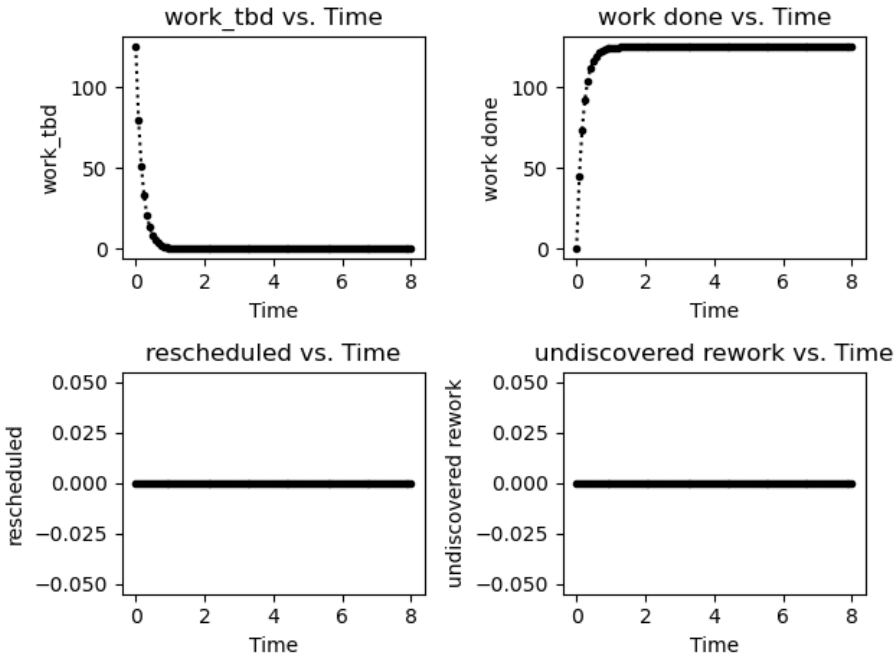


Figure B.107: Walworth Case 0106

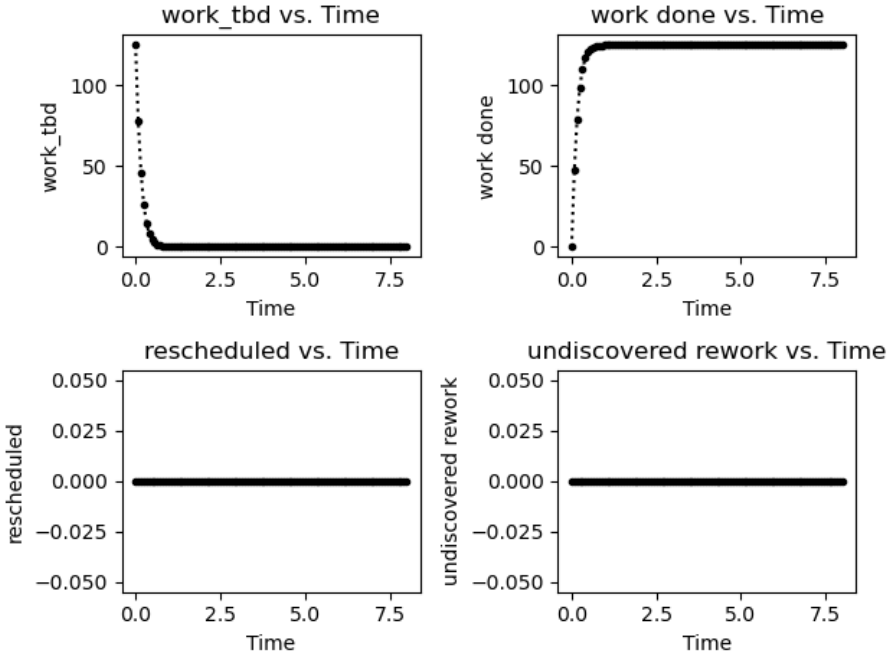


Figure B.108: Walworth Case 0107

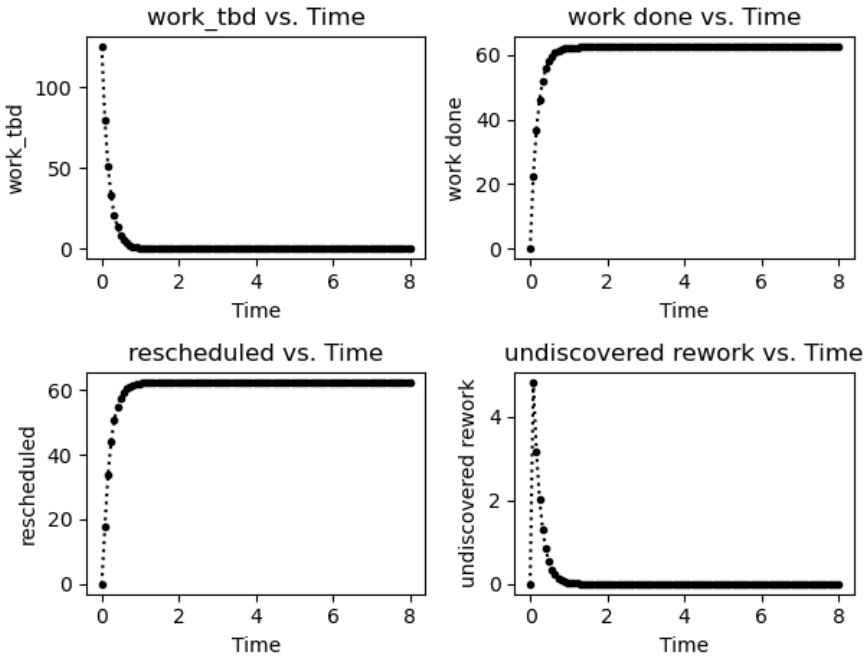


Figure B.109: Walworth Case 0108

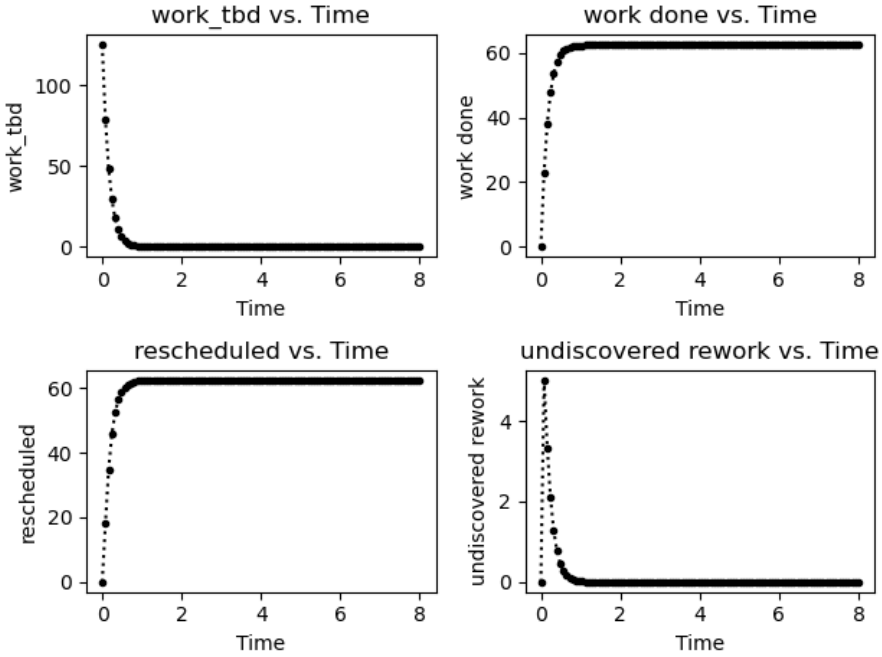


Figure B.110: Walworth Case 0109

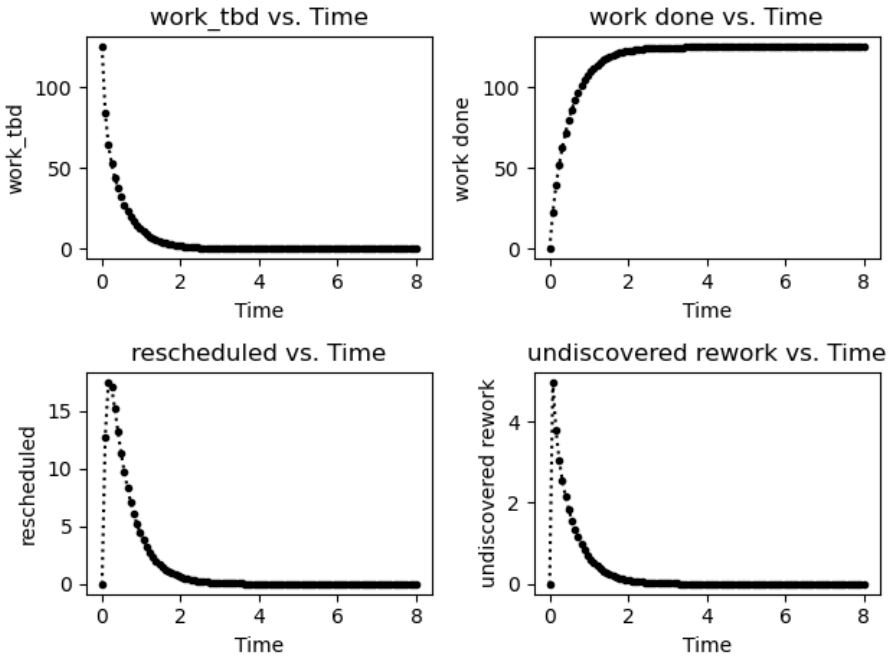


Figure B.111: Walworth Case 0110

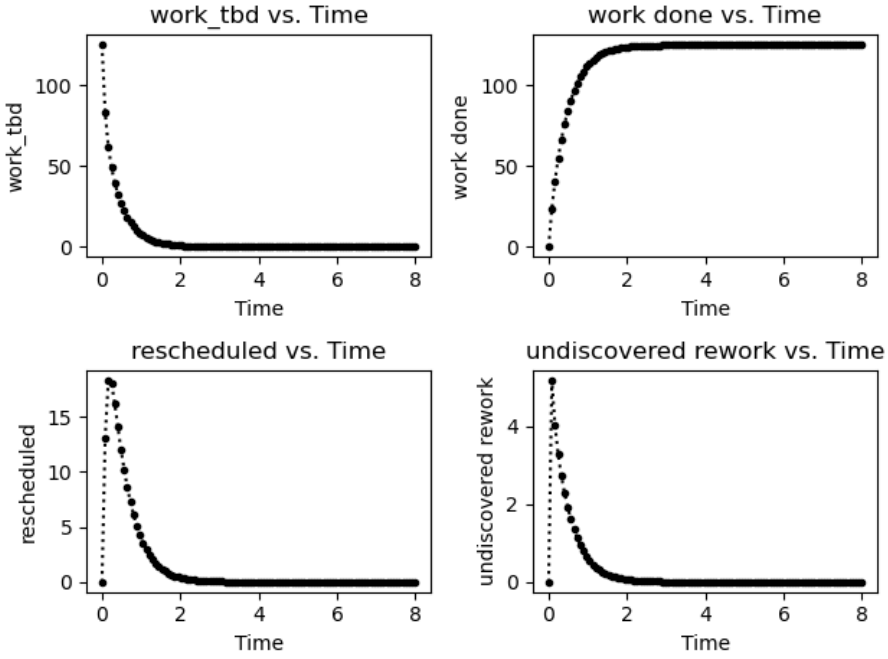


Figure B.112: Walworth Case 0111

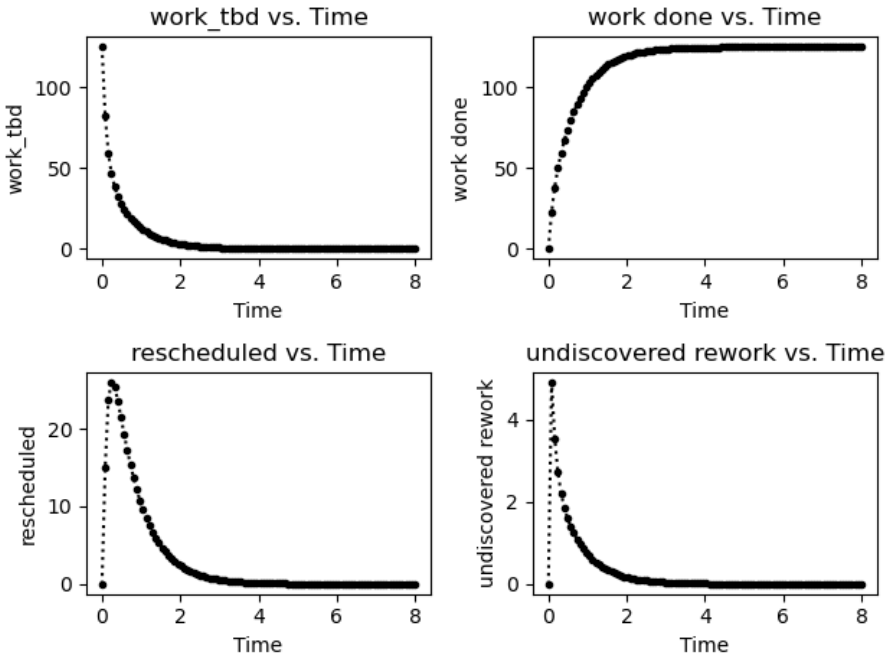


Figure B.113: Walworth Case 0112

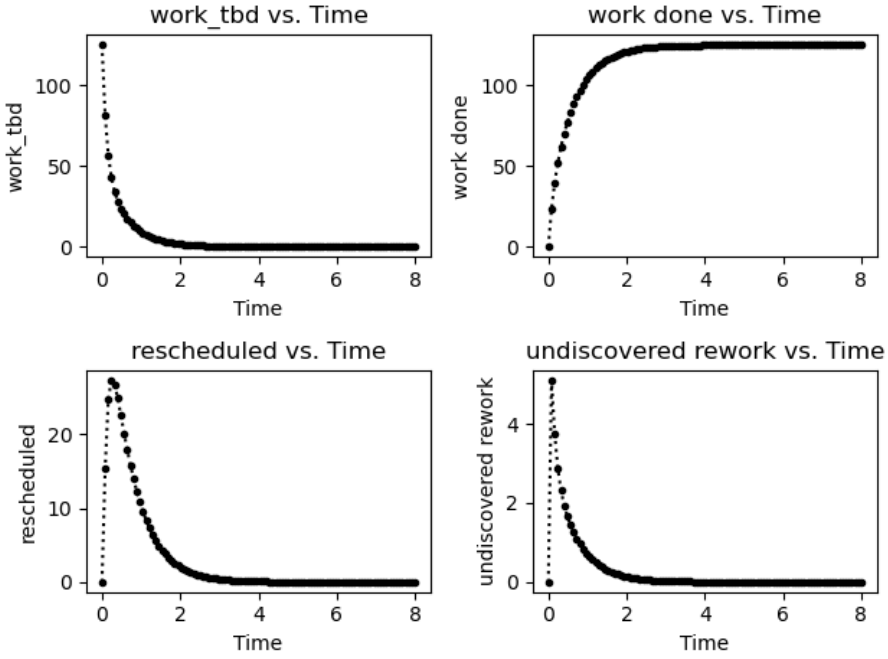


Figure B.114: Walworth Case 0113

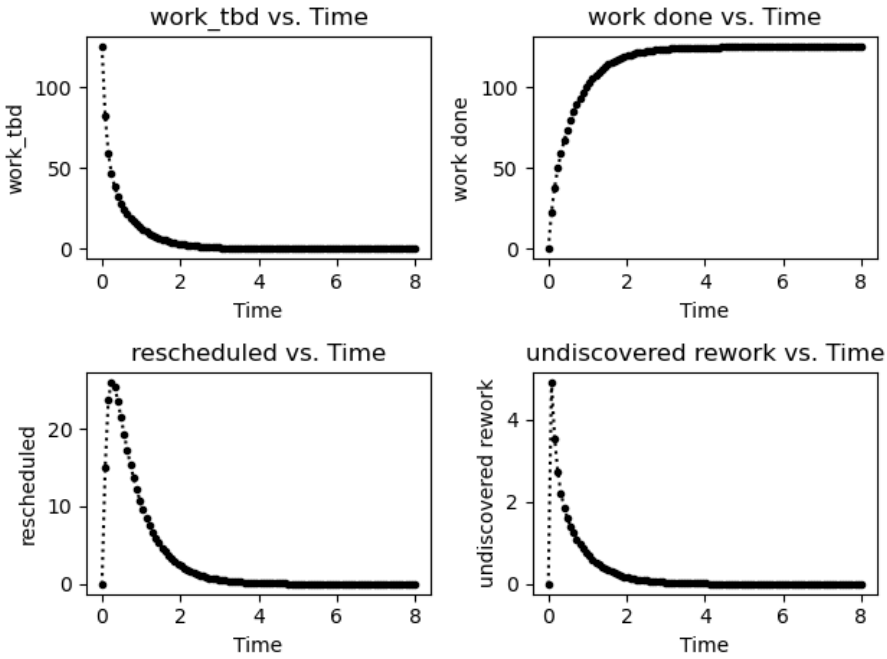


Figure B.115: Walworth Case 0114

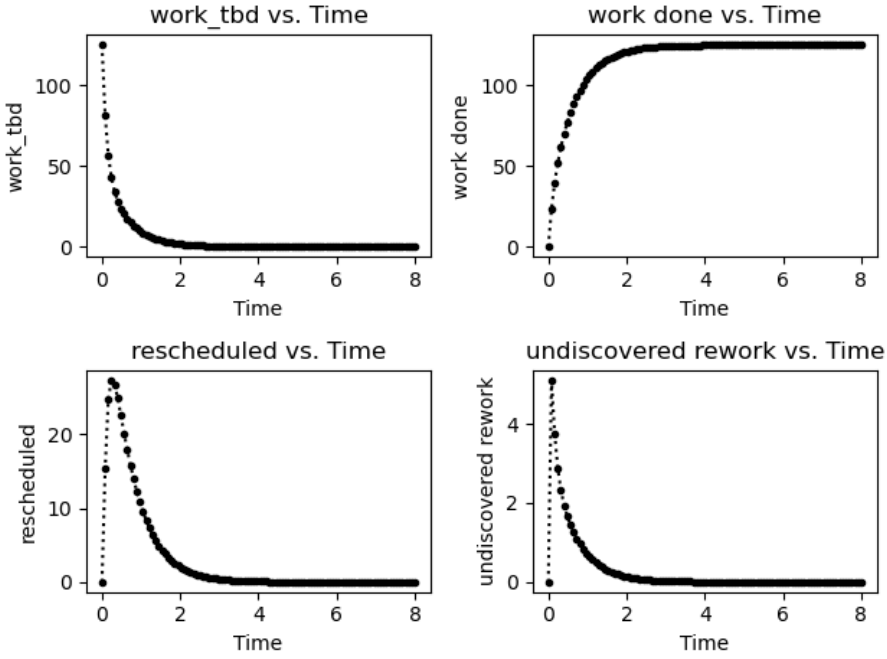


Figure B.116: Walworth Case 0115

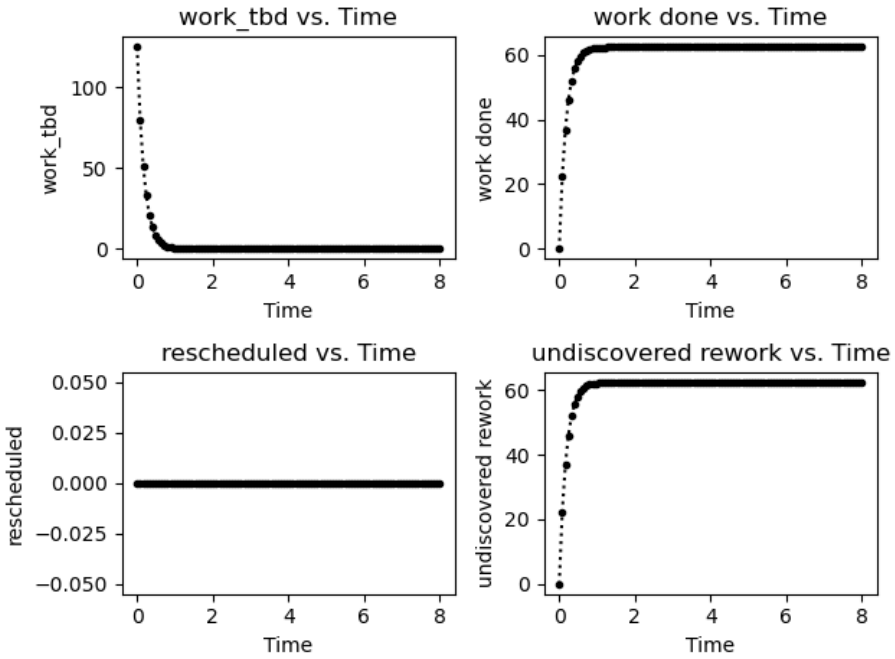


Figure B.117: Walworth Case 0116

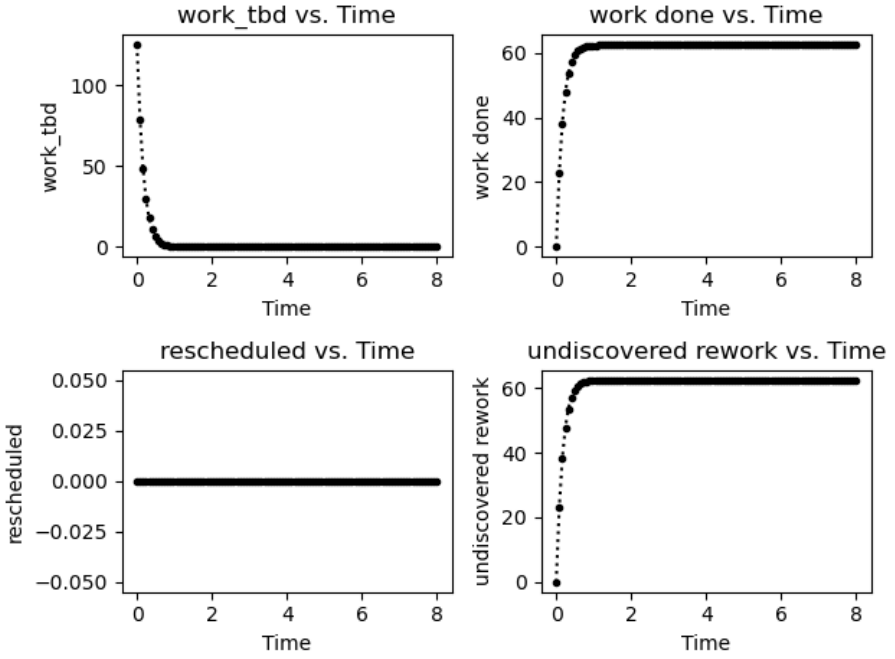


Figure B.118: Walworth Case 0117

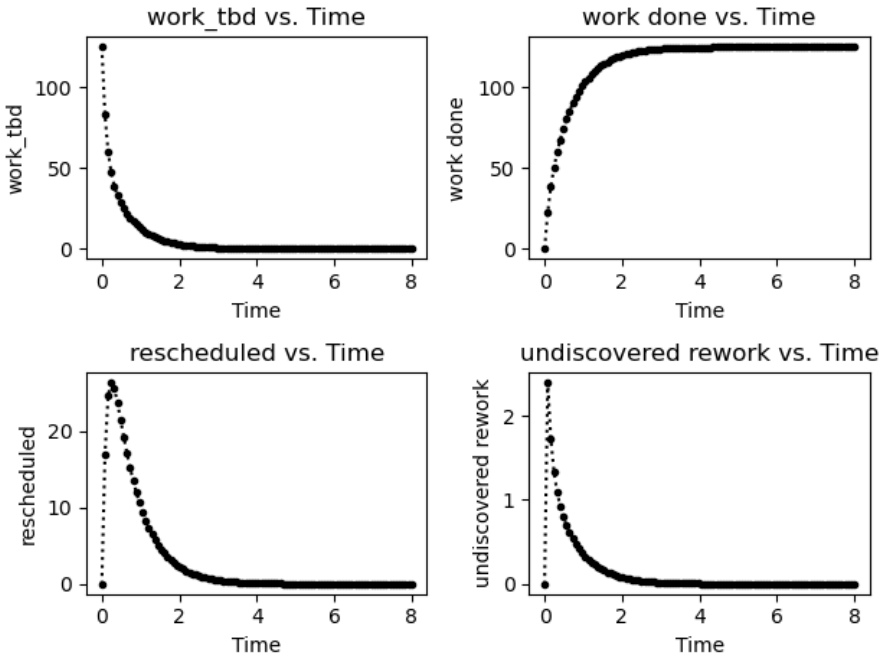


Figure B.119: Walworth Case 0118

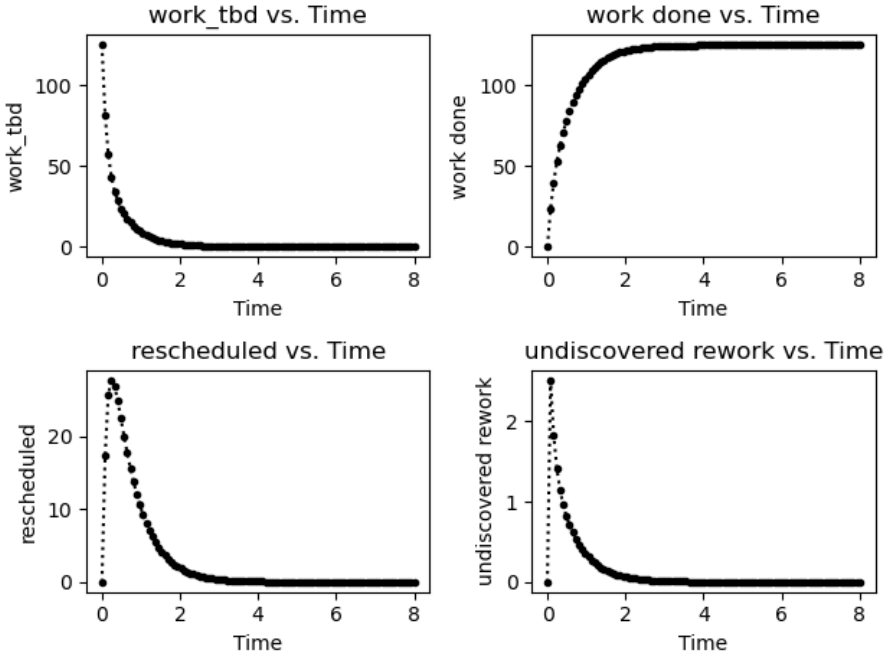


Figure B.120: Walworth Case 0119

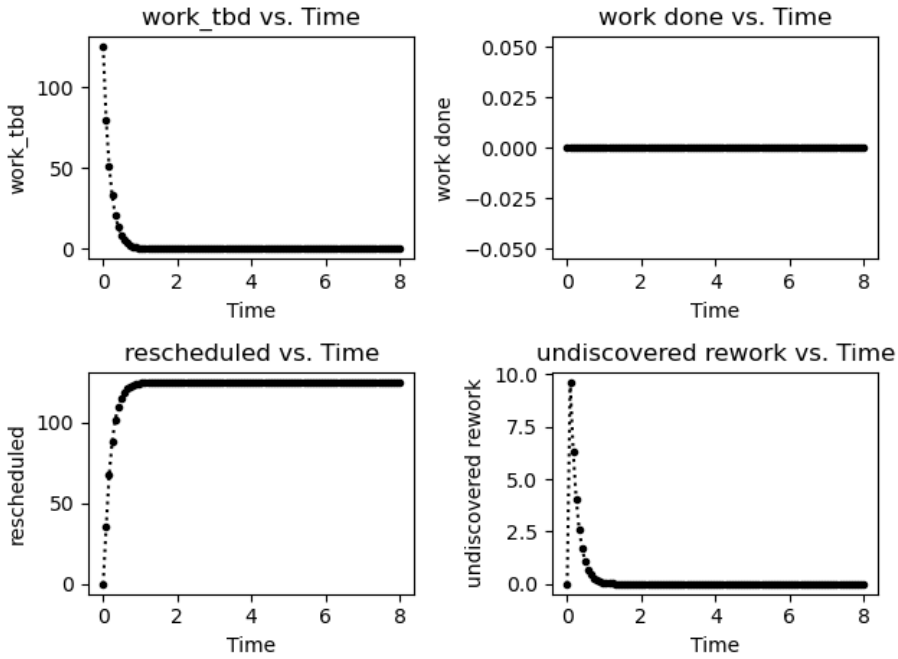


Figure B.121: Walworth Case 0120

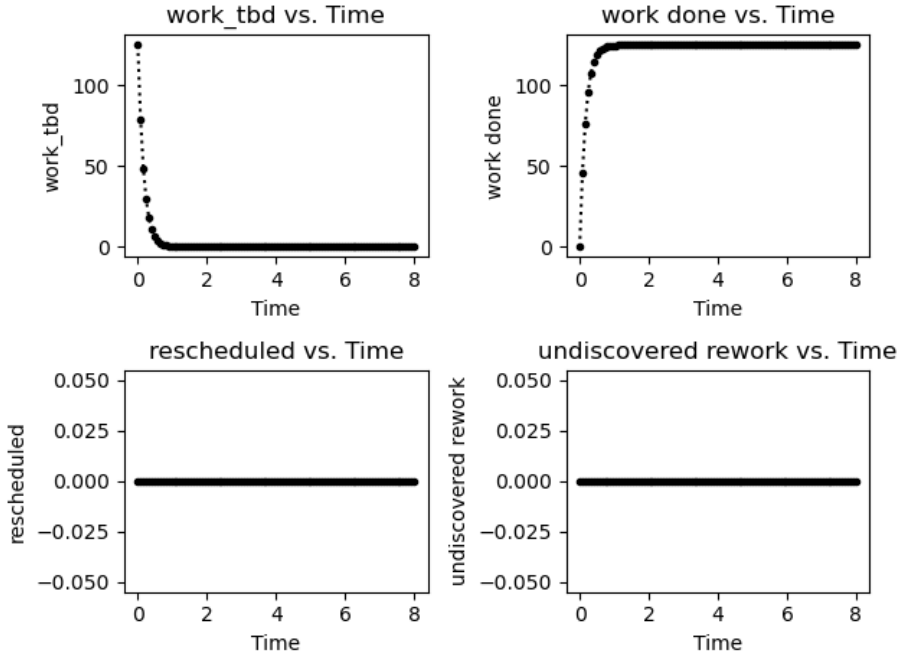


Figure B.122: Walworth Case 0121

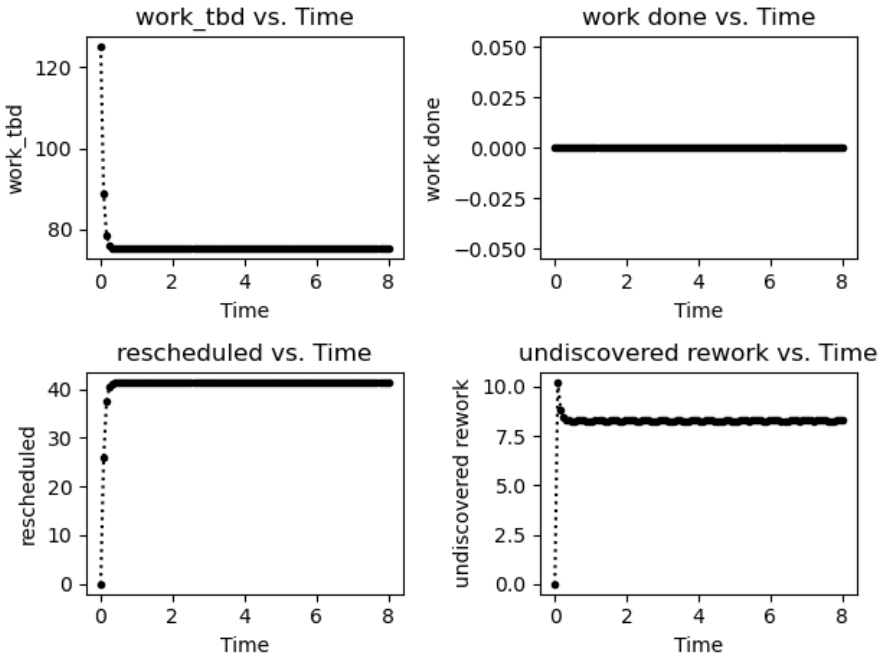


Figure B.123: Walworth Case 0122

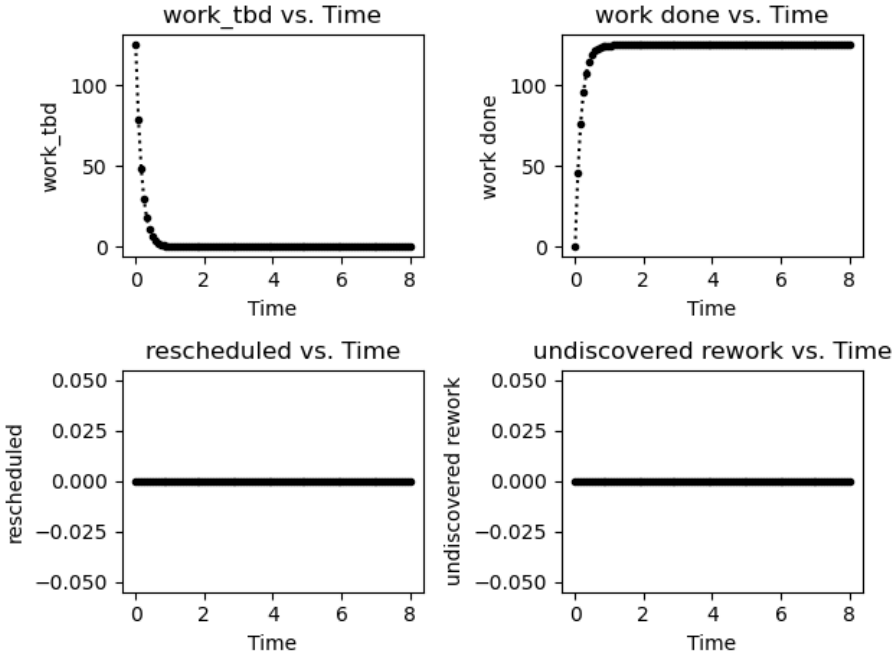


Figure B.124: Walworth Case 0123

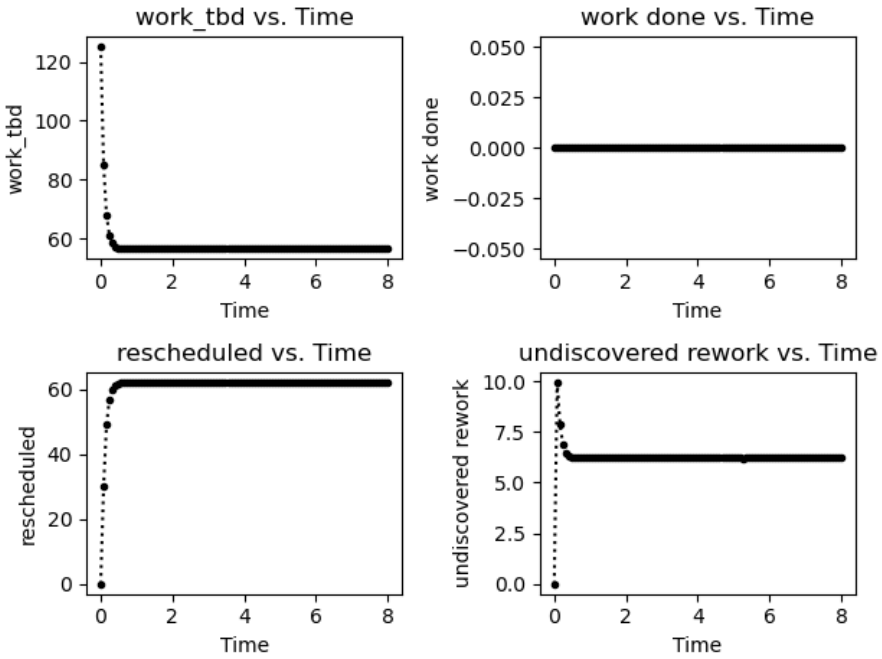


Figure B.125: Walworth Case 0124

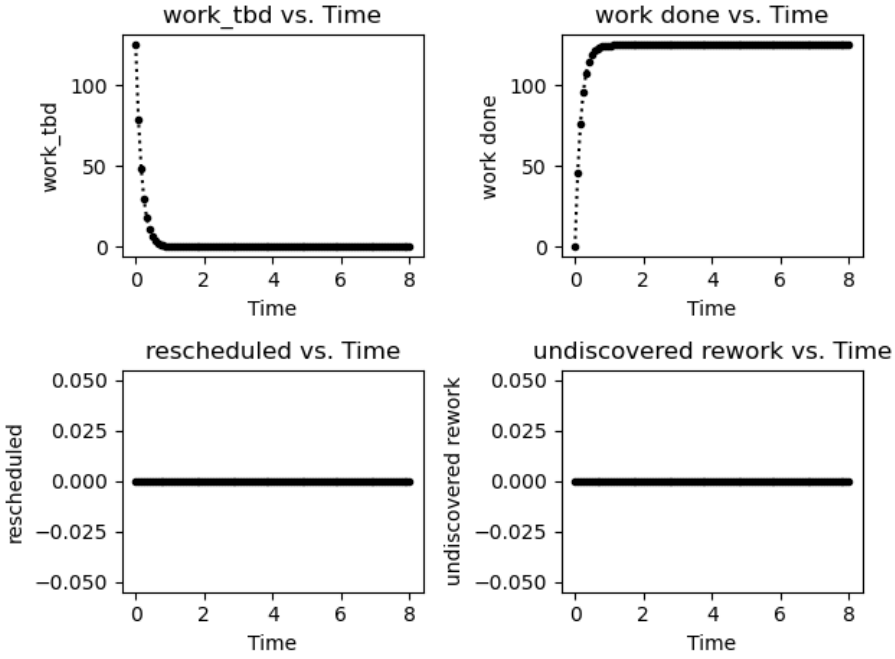


Figure B.126: Walworth Case 0125

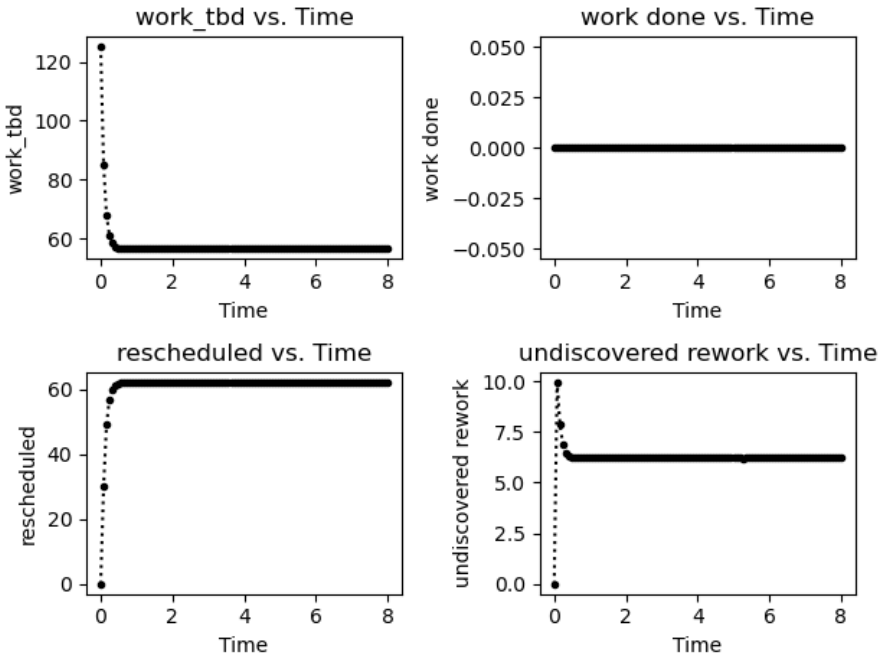


Figure B.127: Walworth Case 0126

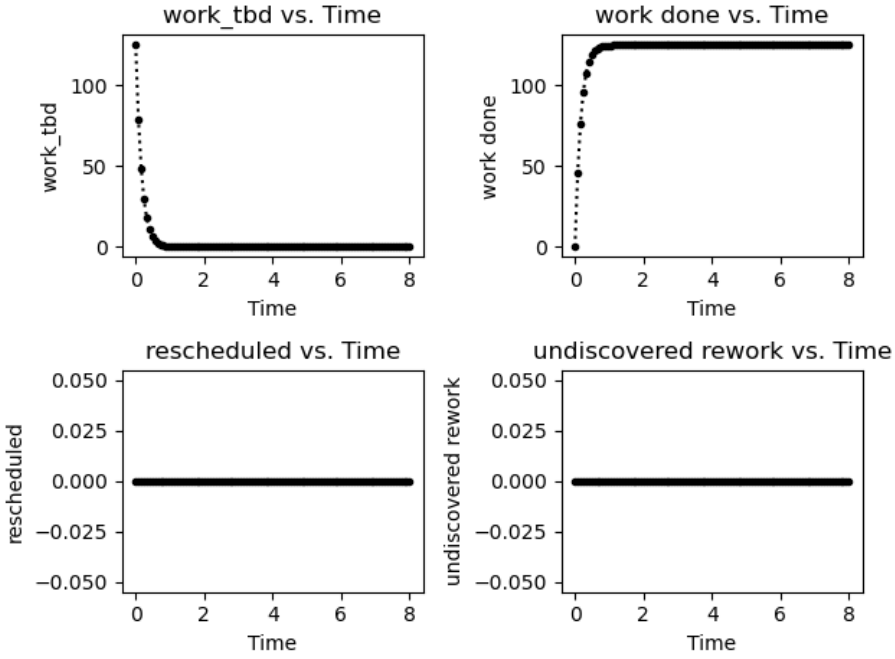


Figure B.128: Walworth Case 0127

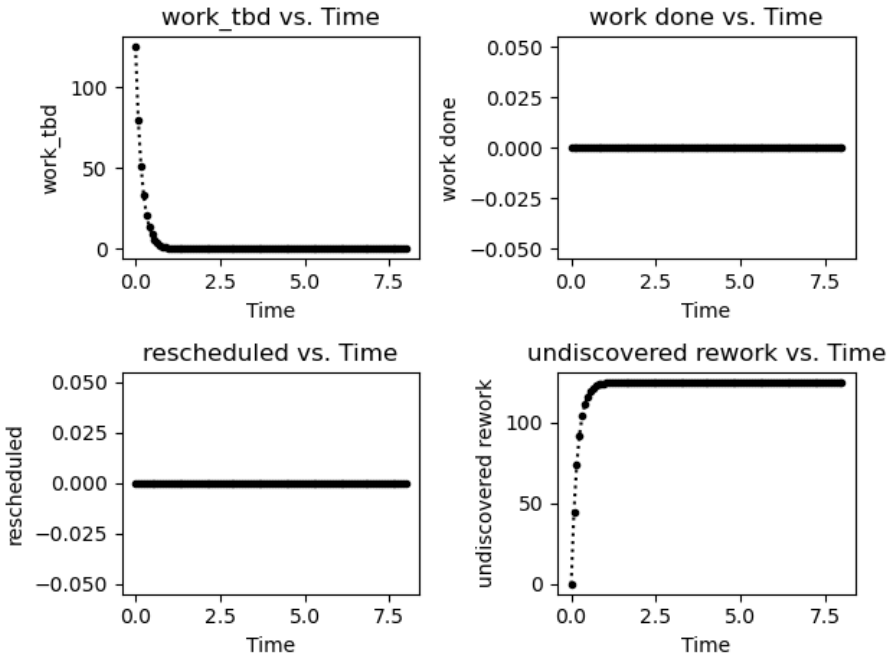


Figure B.129: Walworth Case 0128

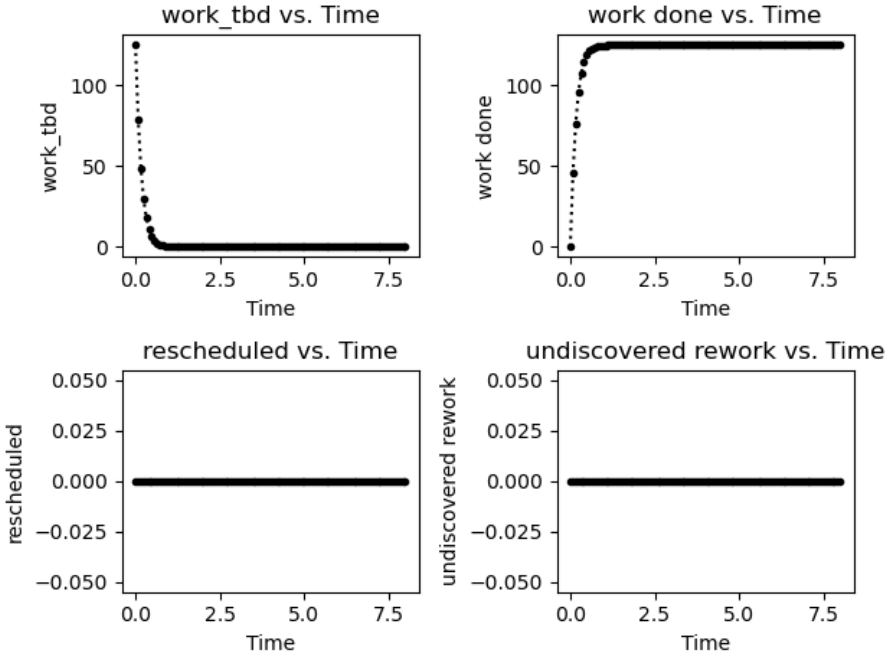


Figure B.130: Walworth Case 0129

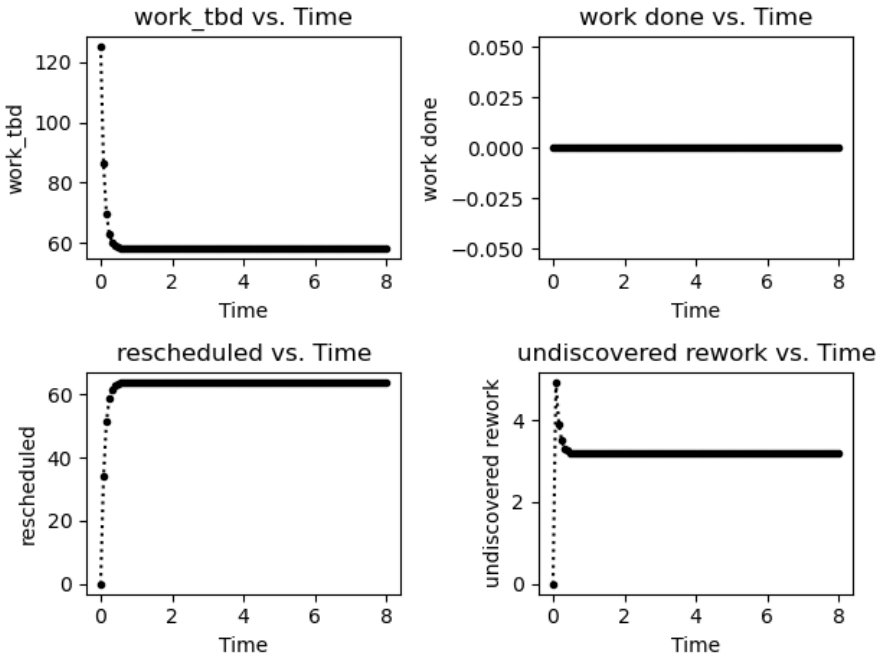


Figure B.131: Walworth Case 0130

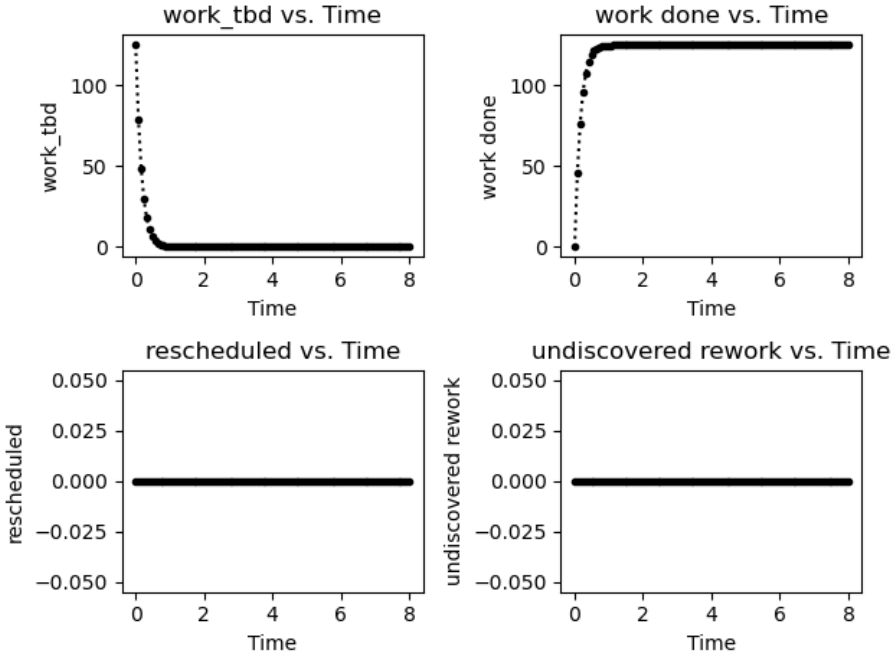


Figure B.132: Walworth Case 0131

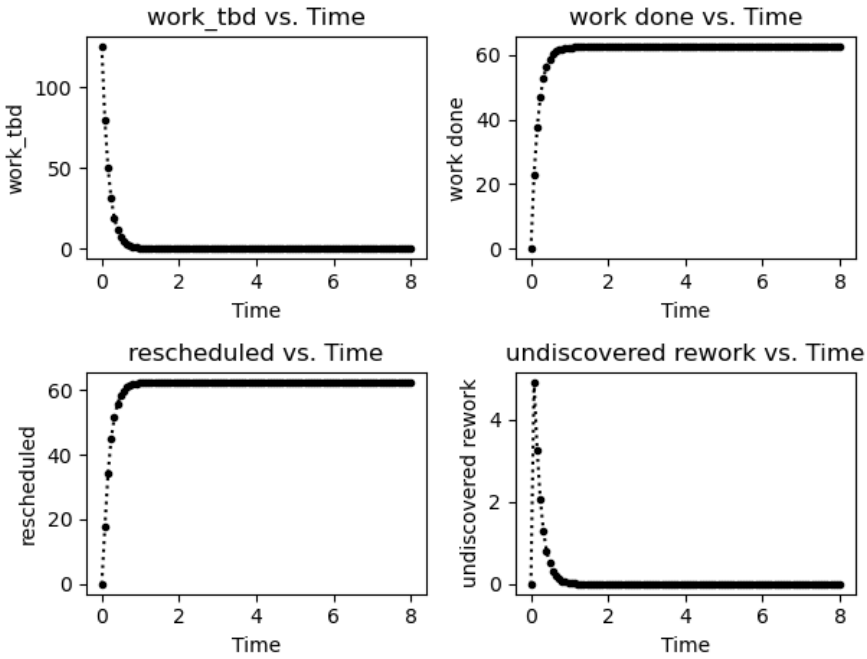


Figure B.133: Walworth Case 0132

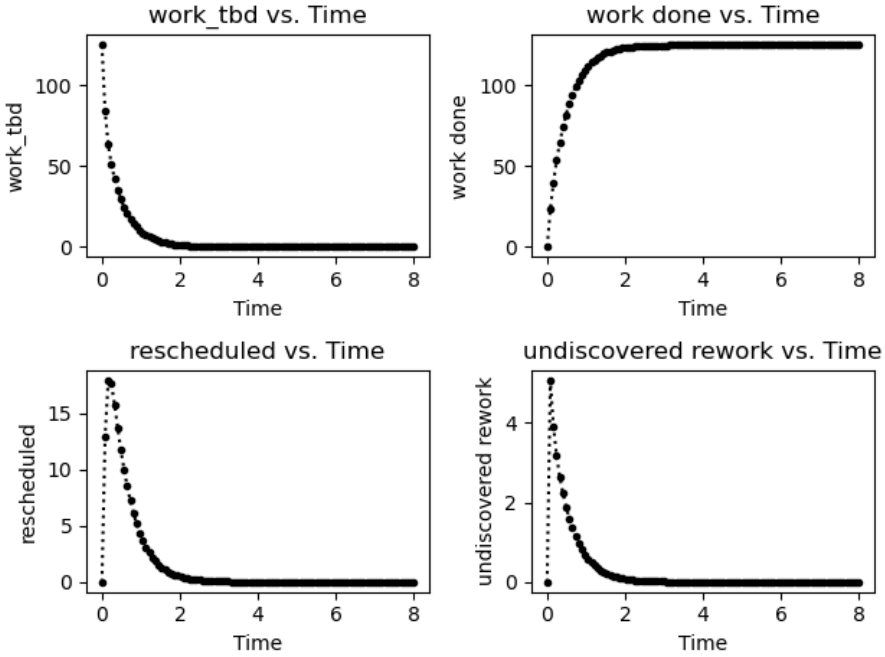


Figure B.134: Walworth Case 0133

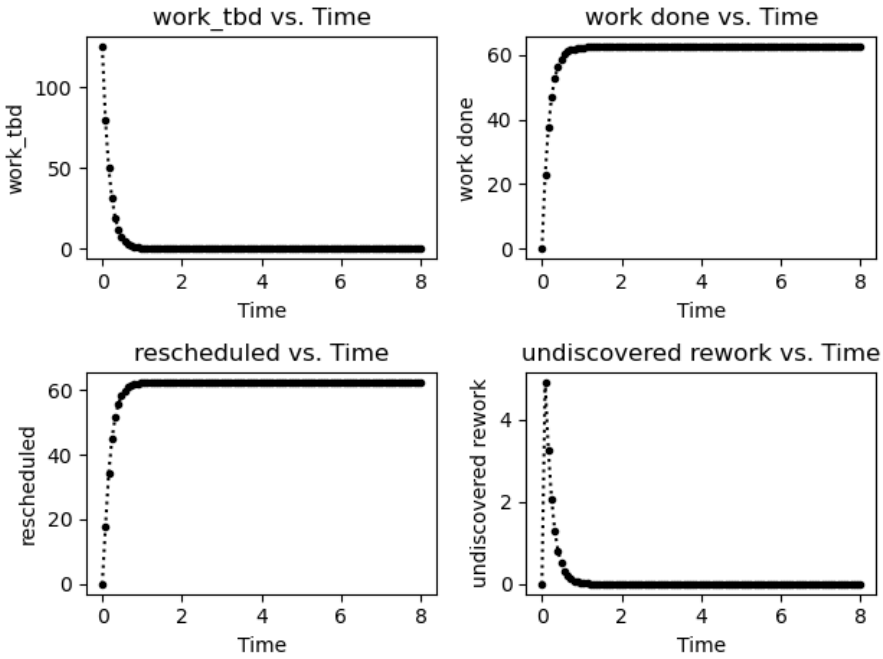


Figure B.135: Walworth Case 0134

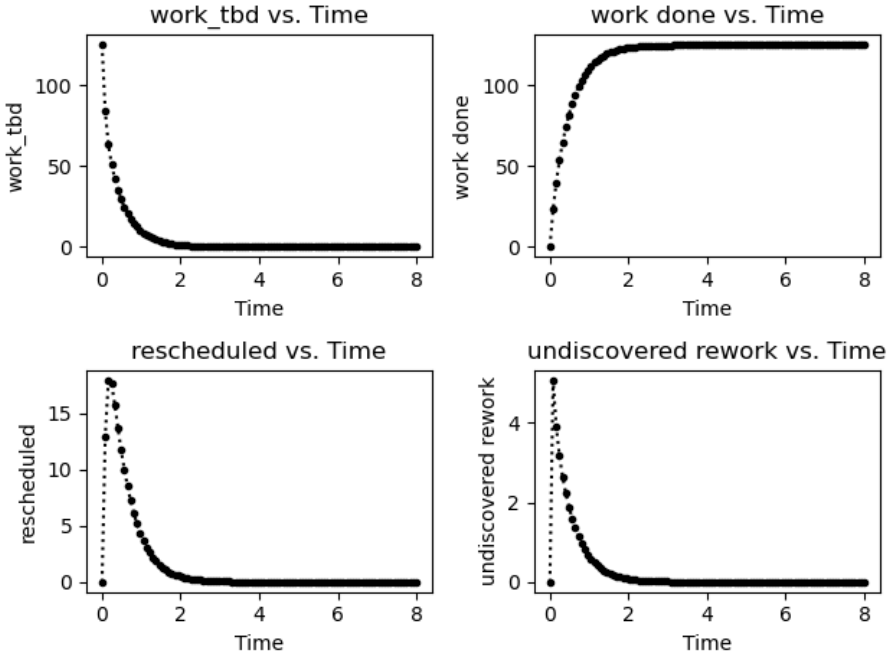


Figure B.136: Walworth Case 0135

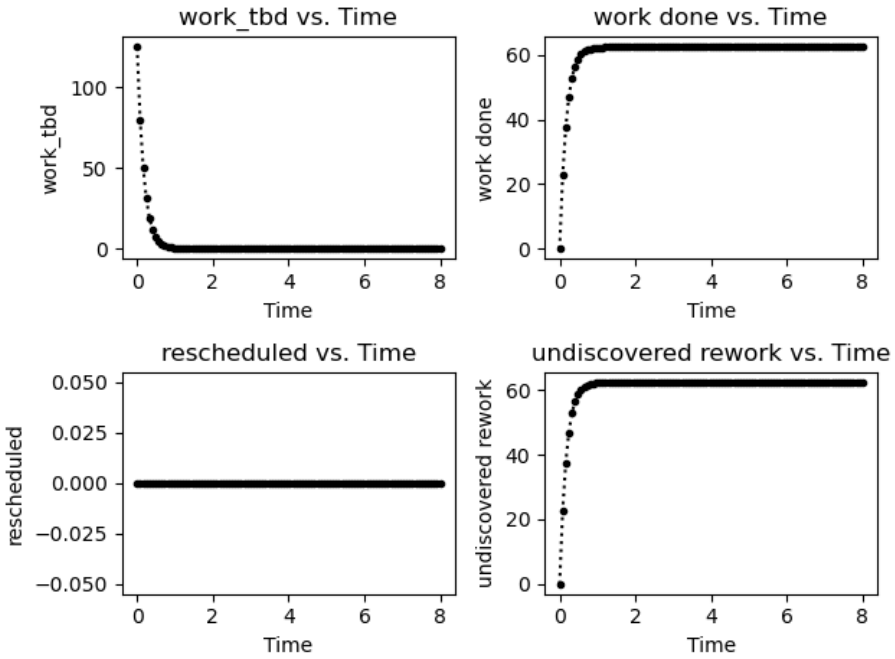


Figure B.137: Walworth Case 0136

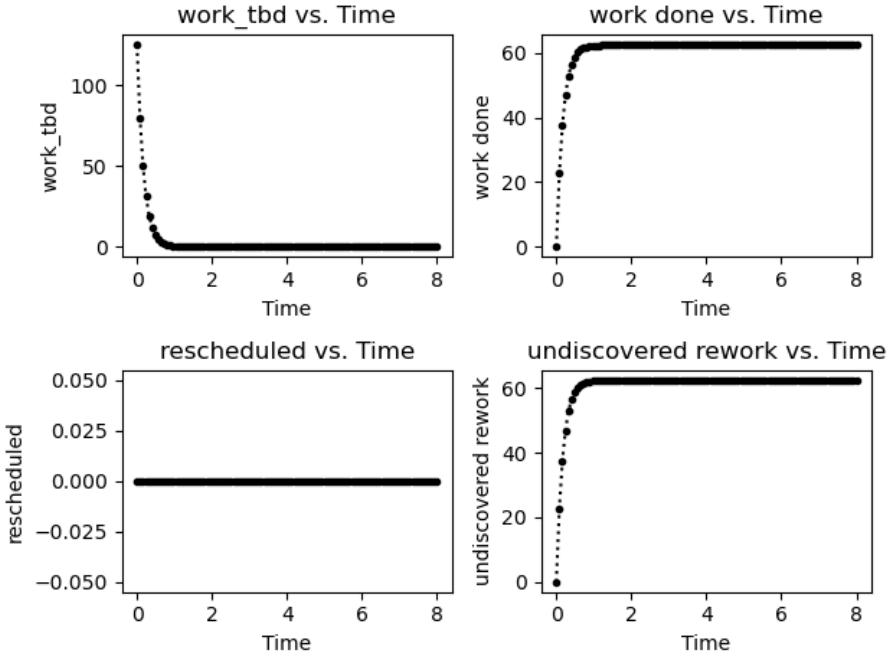


Figure B.138: Walworth Case 0137

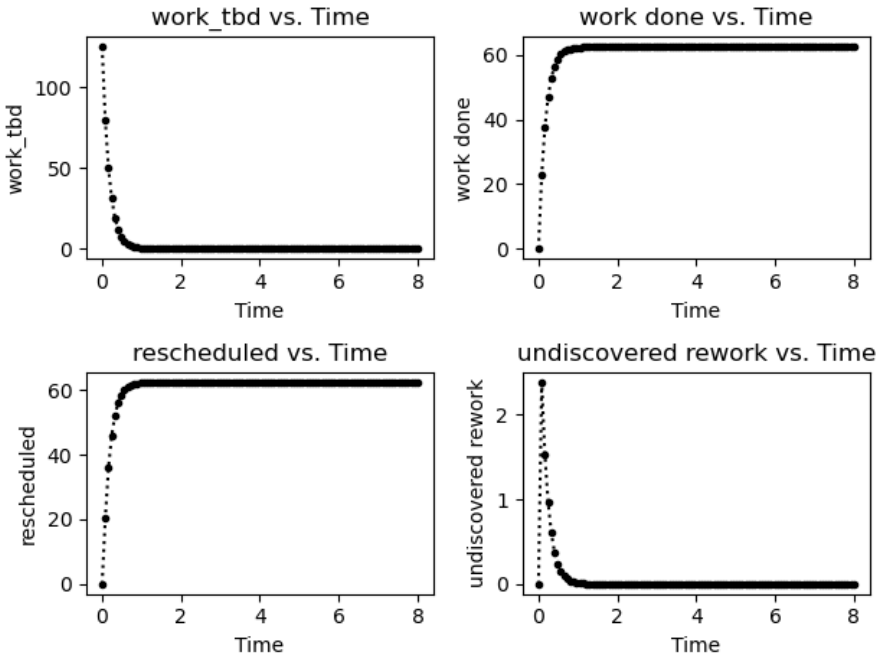


Figure B.139: Walworth Case 0138

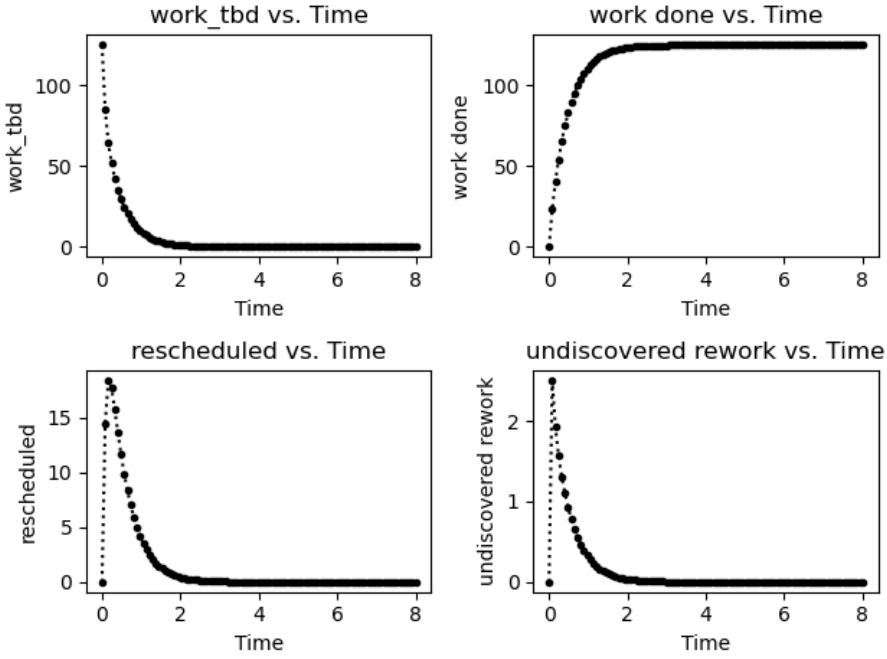


Figure B.140: Walworth Case 0139

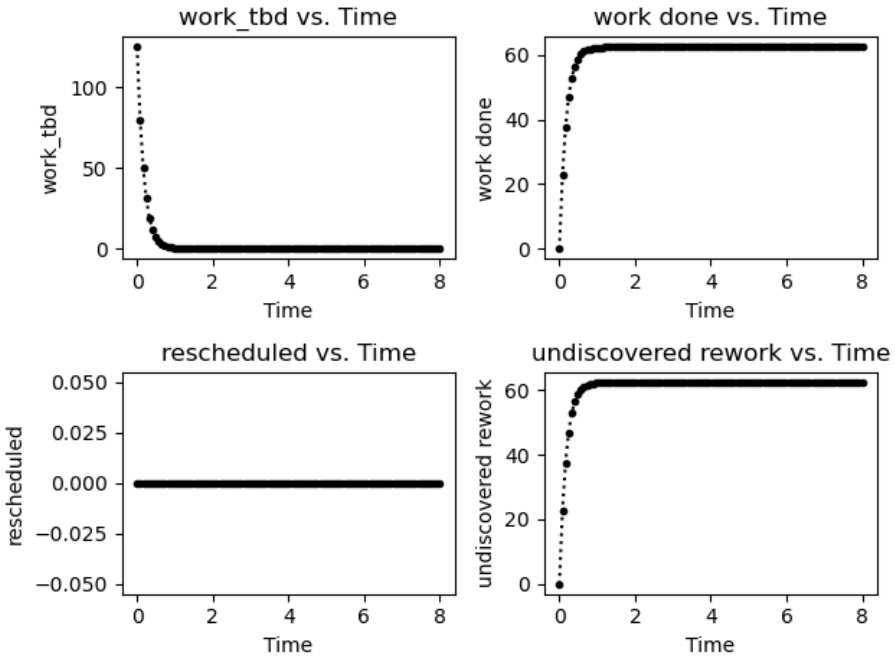


Figure B.141: Walworth Case 0140

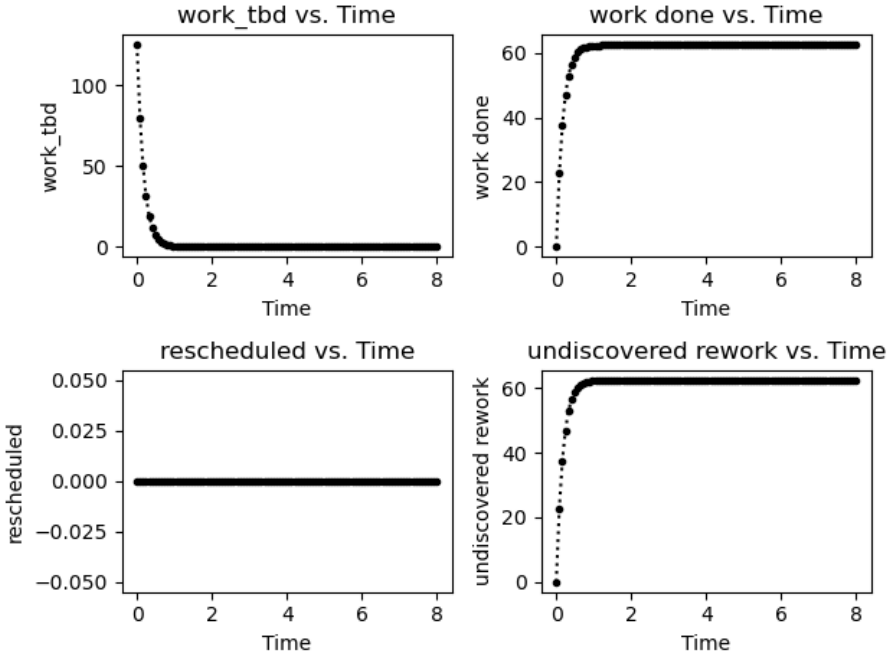


Figure B.142: Walworth Case 0141

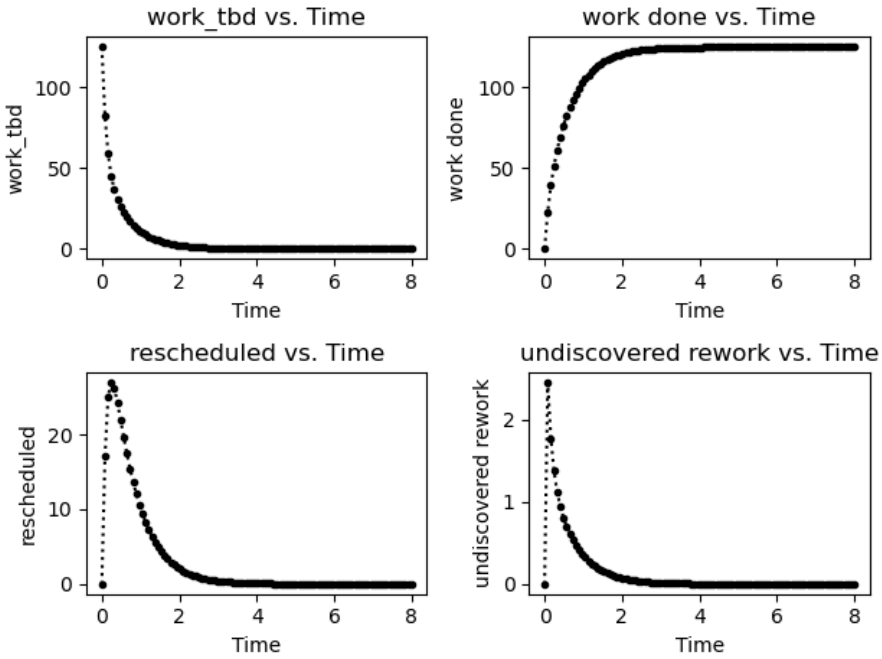


Figure B.143: Walworth Case 0142

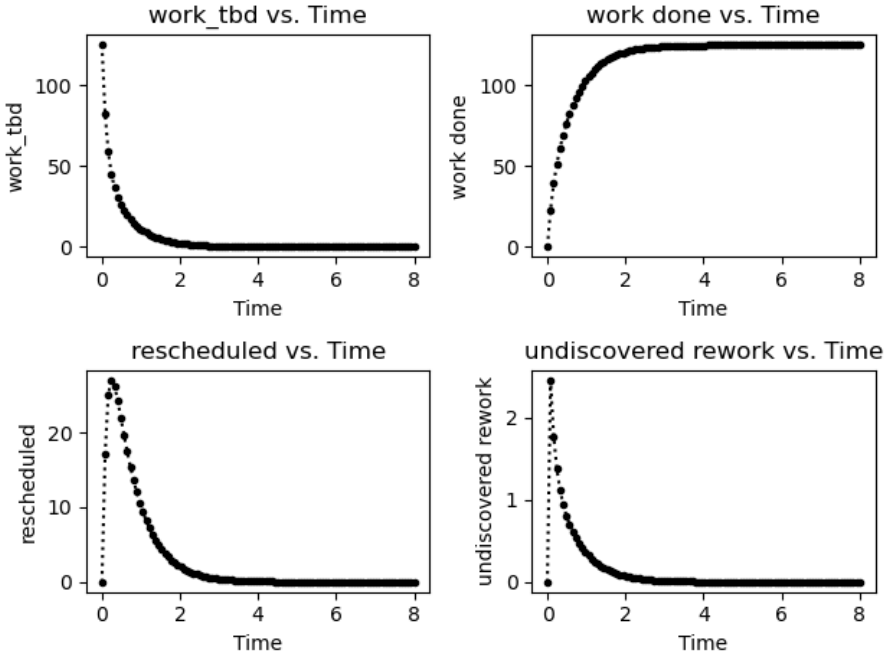


Figure B.144: Walworth Case 0143

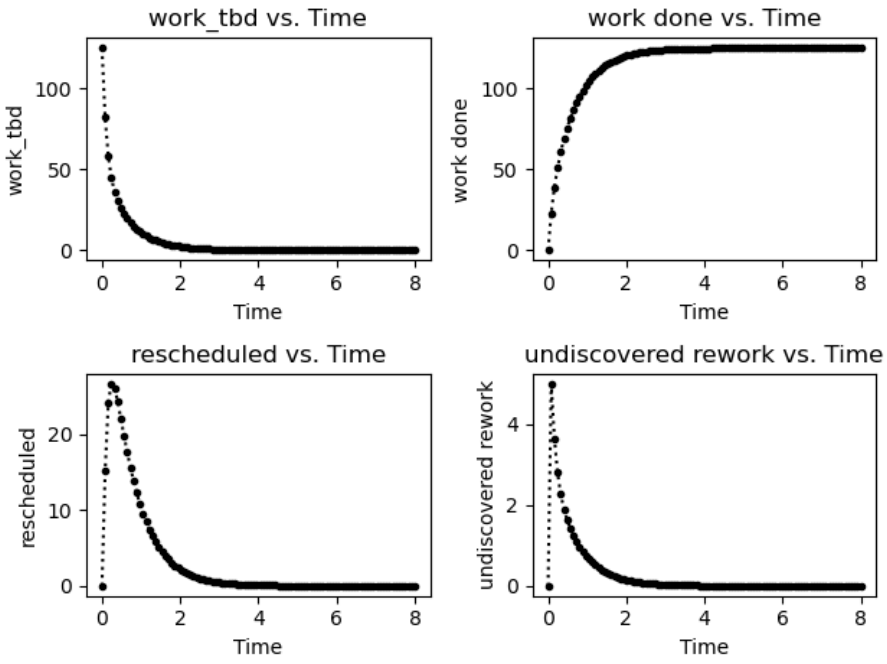


Figure B.145: Walworth Case 0144

APPENDIX C

SCRIPTS SUPPORTING P-SEMP IN MAGICDRAW

Throughout the thesis, several scripts are utilized in MagicDraw.

C.1 Mapping Scripts Established for Walworth Model

Listing C.1: First map from Model to JSON, for Walworth

```
1  // need to handle enums
2  //com.nomagic.uml2.ext.magicdraw.classes.mdkernel.impl.
       EnumerationLiteralImpl
3  import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.*;
4  import com.nomagic.magicdraw.simulation.fuml.fUMLHelper;
5  import fUML.Semantics.Classes.Kernel.Object_;
6  import java.util.ArrayList;
7  import com.google.gson.*;
8
9  def chkbuildres = {elit ->
10     theName = elit.getName()
11     things = []
12     if (elit.getAppliedStereotypeInstance()){
13         for (slot in elit.getAppliedStereotypeInstance().getSlot()){
14             if (slot.hasValue()){
15                 things << slot.getValue().collect{it->it.getValue()
                       ?: it.isValue() ?: it.getInstance().getName() ?:
                       null}
16             }
17         }
18         output = []
19         output << theName
```

```
20          output << things.clone()
21          return output.clone().flatten()
22      }
23      return theName
24  }
25
26  def buildPropMap// define a helper function to assist with looping
        on runtime object
27
28  buildPropMap = {cls,rto ->
29    // Modified property map which gets all owned values of cls from
          RTO
30    def output = [:];// initialize empty output map
31    for (att in cls.getAttribute()){
32      //currVal = ALH.getValue(rto, att.getName())
33      currVal = ALH.getValue(rto,att.getName()) instanceof
            EnumerationLiteral ? ALH.getValue(rto,att.getName()).getName
            () : ALH.getValue(rto,att.getName())
34      if (currVal instanceof ArrayList){
35        def data = []
36        for (itm in currVal){
37                  if (!(att.type.getAppliedStereotypeInstance().
                        getClassifier()[0].getName()=~/Constraint/) && !(
                        att.getAppliedStereotypeInstance().getClassifier
                        ()[0].getName()=~/Port/)){
38                              //print(att.type.
                                    getAppliedStereotypeInstance().
                                    getClassifier()[0].getName())
39                              if (att.type instanceof Class) {
40                                  //println("adding ${itm} to data".
                                        toString())
41                                  data << buildPropMap(att.getType(),
                                        itm);
```

```
42                          //println("data is now ${data}".
                                toString())
43                      }else{
44                          //println("Collecting a value in
                                list...")
45                          currName = att.getName();
46                          //currVal = ALH.getValue(itm,
                                currName);
47                          if (itm instanceof fUML.Semantics.
                                Classes.Kernel.Object_){
48                              currVal = ALH.getValue(itm,
                                    currName) instanceof
                                    EnumerationLiteral ?
                                    chkbuildres(ALH.getValue(itm,
                                    currName)) : ALH.getValue(itm
                                    ,currName)
49                          }else{
50                              currVal = itm
51                          }
52                          //ALH.setValue(rto,currName,currVal
                                +3.14159)// test for changing
                                values.
53                          data << currVal // no loop on ALH
                                methods without the names from
                                the map.
54                      }
55                  }else{
56                      continue
57                  }
58              }
59              //println("data array is ${data}".toString())
60              output[(att.getName())] = data.clone()
61
```

```
62          }else{
63              //println('Found a single value')
64              if (!(att.type.getAppliedStereotypeInstance().
                    getClassifier()[0].getName()=~/Constraint/) && !(att.
                    getAppliedStereotypeInstance().getClassifier()[0].
                    getName()=~/Port/)){
65                  if (att.type instanceof Class) {
66                      output[(att.getName())] = buildPropMap(att.
                            getType(),ALH.getValue(rto,att.getName()));
67                  }else{
68                      //println("Collecting a value...")
69                      currName = att.getName();
70                      //ALH.setValue(rto,currName,currVal+3.14159)//
                            test for changing values.
71                      output[(currName)]=ALH.getValue(rto,att.getName
                            ()) instanceof EnumerationLiteral ?
                            chkbuildres(ALH.getValue(rto,att.getName()))
                            : currVal ; // no loop on ALH methods without
                             the names from the map.
72                  }
73              }else{
74                  continue
75              }
76          }
77      }
78      return output.clone()
79  }
80
81  def getInstanceName = {aRto ->
82    return aRto.toString().split(':')[0].split().join('␣'); //name of
          instance
83  }
84
```

```
 Gson gson = new GsonBuilder()
     .serializeSpecialFloatingPointValues()
     .create();
//print(model_property instanceof java.util.ArrayList)
//print(model_property instanceof fUML.Semantics.Classes.Kernel.
    Object_)
//print(model_property.size())
if (inVar instanceof java.util.ArrayList){
    //print("Parsing array list length ${model_property.size()}")
    obj = control.getTypes()[0]
    //print("Current Object is: ${obj.getName()}")
  itm_map = [:]
  inVar.each{it ->
      //print(it.toString())
    thingmap = buildPropMap(control.getTypes()[0],it)
    //print(thingmap)
    itm_name = getInstanceName(it)
    //thingmap["Feature_Kind"] = control.getTypes()[0].getName()
    itm_map[(itm_name)] = thingmap
  }
  output = gson.toJson(itm_map)
}else if (inVar instanceof fUML.Semantics.Classes.Kernel.Object_){
  thingmap = buildPropMap(control.getTypes()[0],inVar)
  itm_name = getInstanceName(inVar)
  //thingmap["Feature_Kind"] = control.getTypes()[0].getName()
  output = gson.toJson([(itm_name):thingmap])
}else{
  output = 'Error! Inappropriate use of ParseArgsToJson'
}
```

Listing C.2: First map from JSON to Model, for Walworth

```
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.*;
```

```
2  import com.nomagic.magicdraw.simulation.fuml.fUMLHelper;
3  import fUML.Semantics.Classes.Kernel.Object_;
4  import java.util.ArrayList;
5  import com.google.gson.*;
6
7  def chksetres = {theObj, vname, vval ->
8      // handle enumeration literal setting (w/ tag list)
9      //
10     //print(theObj.toString())
11     //print(vname)
12     //print(vval)
13     currVal = ALH.getValue(theObj, vname)
14     if (currVal instanceof EnumerationLiteral){
15         if (vval instanceof List){
16             ALH.setValue(theObj, vname, vval[0])
17         } else {
18             ALH.setValue(theObj, vname, vval)
19         }
20     } else {
21         ALH.setValue(theObj, vname, vval)
22     }
23 }
24
25
26 def buildPropMap// define a helper function to assist with looping
       on runtime object
27
28 buildPropMap = {cls,rto, setterFlag=false, setterObj=null ->
29     def output = [:];// output map.
30     if (!setterFlag || setterObj==null){
31         return output
32     }
33   println("Beginning␣loop␣for␣${cls.getName()}".toString())  //
```

```
        debug //print statements.
34    println("The runtime object is: ${rto.toString()}".toString())
35      for (att in cls.getAttribute()){// find the names according to
            the MD API, and save to map.
36          ////println("output is currently: ${output}".toString())
37          println("The attribute is: ${att.getName()}".toString())
38          ////println("The attribute type is: ${att.type}".toString())
39          println("The runtime property is: ${ALH.getValue(rto,att.
                getName())}".toString())
40          //if(setterFlag){println("the setter value is: ${setterObj[(
                att.getName().toString())]}".toString())}
41          currVal = ALH.getValue(rto,att.getName())
42          setVal = setterObj[(att.getName())]
43          if (currVal instanceof ArrayList){
44              println('Found a list')
45              def data = []
46              for (itm in currVal){
47                  if (!(att.type.getAppliedStereotypeInstance().
                        getClassifier()[0].getName()=~/Constraint/)){
48                              if (att.type instanceof Class) {
49                                  data << buildPropMap(att.getType(),
                                        itm, setterFlag,setterObj[(att.
                                        getName())]);
50                              }else{
51                                  println("Collecting a value in list
                                        ...")
52                                  currName = att.getName();
53                                  //currVal = ALH.getValue(itm,
                                        currName);
54                                  //ALH.setValue(rto,currName,currVal
                                        +3.14159)// test for changing
                                        values.
55                                  if(setterFlag){
```

```
56                                      chksetres(rto,currName,setterObj
                                            [(currName)])
57                                  }
58                                  data << 0; // no loop on ALH methods
                                        without the names from the map.
59                              }
60                          }else{
61                              continue
62                          }
63                      }
64                      output[(att.getName())] = data.clone()
65
66          }else{
67              println('Found a single value')
68              if (!(att.type.getAppliedStereotypeInstance().
                    getClassifier()[0].getName()=~/Constraint/)){
69                  print(att.type)
70                  if (att.type instanceof Class) {
71                      output[(att.getName())] = buildPropMap(att.
                            getType(),ALH.getValue(rto,att.getName()),
                            setterFlag,setterObj[(att.getName())]);
72                  }else{
73                      println("Collecting a value...")
74                      currName = att.getName();
75                      if (setterFlag){
76                          print(currName)
77                          print(setterObj)
78                          chksetres(rto,currName,setterObj[(currName)
                                ])
79                          output[(currName)]=setterObj[(currName)]
80                      }else{
81                          //ALH.setValue(rto,currName,currVal+3.14159)
                                // test for changing values.
```

```
82                          output[(currName)]=[currVal]; // no loop on
                                ALH methods without the names from the
                                map.
83                      }
84                  }
85              }else{
86                  continue
87              }
88          }
89      }
90      return output.clone()
91  }
92
93  def getInstanceName = {aRto ->
94    return aRto.toString().split(':')[0].split().join('␣'); //name of
          instance
95  }
96
97   Gson gson = new GsonBuilder()
98       .serializeSpecialFloatingPointValues()
99       .create();
100
101 def resmap = gson.fromJson(res,Map.class)
102 print(resmap)
103 def resKeySet = resmap.keySet()
104 print(model_property instanceof java.util.ArrayList)
105 print(model_property instanceof fUML.Semantics.Classes.Kernel.
        Object_)
106 //print(model_property.size())
107 def obj = model_obj.getTypes()[0]
108 if (model_property instanceof java.util.ArrayList){
109     print("Parsing␣array␣list␣length␣${model_property.size()}")
110     print("Current␣Object␣is:␣${obj.getName()}")
```

```
111   model_property.each{it ->
112       print(it.toString())
113       itm_name = getInstanceName(it)
114       if (resKeySet.contains(itm_name)) {
115          curr_data = resmap[(itm_name)]
116          somedatares = buildPropMap(obj,it,true,curr_data)
117       }
118   }
119 }else if (model_property instanceof fUML.Semantics.Classes.Kernel.
       Object_){
120   somedatares = buildPropMap(obj,model_property,true,resmap)
121   /*
122   itm_name = getInstanceName(model_property)
123   if (resKeySet.contains(itm_name)) {
124          curr_data = resmap[(itm_name)]
125          somedatares = buildPropMap(obj,model_property,true,
               curr_data)
126       }
127   */
128 }else{
129   println('Error!')
130 }
```

C.2 Mapping Scripts Established with the DES DSL Profile

Listing C.3: Mapping Process Node Content to JSON stored in Action List

```
1 import com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
2 import com.google.gson.*;
3 import groovy.json.*;
4 def jsonslurper = new JsonSlurper(type: JsonParserType.LAX)
5 Gson gson = new Gson()
6 res = []
```

```
7  sortedActions = arg1.sort{it->
8      if(StereotypesHelper.hasStereotype(it)){
9          valList = StereotypesHelper.getStereotypePropertyValue(it, '
               ProcessAction', 'no')
10         return (valList.get(0) as Integer)
11     }
12 }
13 var = null
14 sortedActions.each{it->
15     if (!(StereotypesHelper.getStereotypes(it).collect{ti->ti.
           getName()}.get(0)==~'ConditionalAction')){
16         res<<[(it.getName()):it.actionBody]
17         var = StereotypesHelper.getStereotypes(it).collect{ti->ti.
               getName()}
18     }else{
19         thisMap = gson.fromJson(it.actionBody,Map.class)
20         res<<[(it.getName()):thisMap]
21     }
22 }
23 JsonOutput.toJson(res)
```

Listing C.4: Mapping Conditional Action Content to JSON stored in Condition Description

```
1  import com.nomagic.uml2.ext.magicdraw.actions.mdbasicactions.
       OpaqueAction;
2  import com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
3  import com.google.gson.*;
4  import groovy.json.*;
5
6  def theRes = [test:[:],iftrue:[],iffalse:[]]
7
8  clauses = arg1.sort{it.hasTest() ? it.getTest() : null}
```

```
for (aclause in clauses){
    if (aclause.hasTest()){
        alltests = aclause.getTest()
        alltests.each{it ->
            if (it instanceof OpaqueAction){
                if (it.hasBody()){
                    bodyContents = it.getBody()
                    bodyText = bodyContents[0]
                    bodyTextList = bodyText.split(',')
                    inner = [(bodyTextList[1]):bodyTextList[2] as
                        Integer]
                    theRes.test << [(bodyTextList[0]):inner]
                }
            }
        }
        if (aclause.hasBody()){
            allBodyActions = aclause.getBody()
            justProcessActions = allBodyActions.findAll{(
                StereotypesHelper.getStereotypes(it).collect{ti->ti.
                getName()}.get(0)==~'ProcessAction')}//findAll .
                toString()==~'ProcessAction').find()}
            sortedProcessActions = justProcessActions.clone()//sort{
                it.actionNumbering}
            sortedProcessActions.each{it->theRes.iftrue << [(it.
                getName()):it.actionBody]}
        }
    }else{
        if (aclause.hasBody()){
            allBodyActions = aclause.getBody()
            justProcessActions = allBodyActions.findAll{(
                StereotypesHelper.getStereotypes(it).collect{ti->ti.
                getName()}.get(0)==~'ProcessAction')}//findAll .
                toString()==~'ProcessAction').find()}
```

```
33              sortedProcessActions = justProcessActions.clone()//sort{
                    it.actionNumbering}
34              sortedProcessActions.each{it->theRes.iffalse << [(it.
                    getName()): (it.getName() =='yieldtime' ? it.
                    actionBody as Integer : it.actionBody)]}
35          }
36      }
37  }
38  //theRes.toString()
39  Gson gson = new Gson()
40  JsonOutput.toJson(theRes)
```

C.3 Revised Mapping Script for Repeated Use in RDS Interface

Listing C.5: Second map from Model to JSON, for RDS Interface

```
1   // need to handle enums
2   //com.nomagic.uml2.ext.magicdraw.classes.mdkernel.impl.
        EnumerationLiteralImpl
3   import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.*;
4   import com.nomagic.magicdraw.simulation.fuml.fUMLHelper;
5   import fUML.Semantics.Classes.Kernel.Object_;
6   import java.util.ArrayList;
7   import com.google.gson.*;
8
9   def chkbuildres = {elit ->
10      theName = elit.getName()
11      things = []
12      if (elit.getAppliedStereotypeInstance()){
13          for (slot in elit.getAppliedStereotypeInstance().getSlot()){
14              if (slot.hasValue()){
15                  things << slot.getValue().collect{it->it.getValue()
                        ?: it.isValue() ?: it.getInstance().getName() ?:
```

```
                    null}
16              }
17          }
18          output = []
19          output << theName
20          output << things.clone()
21          return output.clone().flatten()
22      }
23      return theName
24  }
25
26  def buildPropMap// define a helper function to assist with looping
        on runtime object
27
28  buildPropMap = {cls,rto ->
29    // Modified property map which gets all owned values of cls from
          RTO
30    def output = [:];// initialize empty output map
31    for (att in cls.getAttribute()){
32      //currVal = ALH.getValue(rto, att.getName())
33      currVal = ALH.getValue(rto,att.getName()) instanceof
            EnumerationLiteral ? ALH.getValue(rto,att.getName()).getName
            () : ALH.getValue(rto,att.getName())
34      if (currVal instanceof ArrayList){
35        def data = []
36        for (itm in currVal){
37                  if (!(att.type.getAppliedStereotypeInstance().
                        getClassifier()[0].getName()=~/Constraint/) && !(
                        att.getAppliedStereotypeInstance().getClassifier
                        ()[0].getName()=~/Port/)){
38                              //print(att.type.
                                    getAppliedStereotypeInstance().
                                    getClassifier()[0].getName())
```

```
39                      if (att.type instanceof Class) {
40                          //println("adding ${itm} to data".
                                toString())
41                          data << buildPropMap(att.getType(),
                                itm);
42                          //println("data is now ${data}".
                                toString())
43                      }else{
44                          //println("Collecting a value in
                                list...")
45                          currName = att.getName();
46                          //currVal = ALH.getValue(itm,
                                currName);
47                          if (itm instanceof fUML.Semantics.
                                Classes.Kernel.Object_){
48                              currVal = ALH.getValue(itm,
                                    currName) instanceof
                                    EnumerationLiteral ?
                                    chkbuildres(ALH.getValue(itm,
                                    currName)) : ALH.getValue(itm
                                    ,currName)
49                          }else{
50                              currVal = itm
51                          }
52                          //ALH.setValue(rto,currName,currVal
                                +3.14159)// test for changing
                                values.
53                          data << currVal // no loop on ALH
                                methods without the names from
                                the map.
54                      }
55                  }else{
56                      continue
```

```
57                          }
58                      }
59                      //println("data array is ${data}".toString())
60                      output[(att.getName())] = data.clone()
61
62          }else{
63              //println('Found a single value')
64              if (!(att.type.getAppliedStereotypeInstance().
                    getClassifier()[0].getName()=~/Constraint/) && !(att.
                    getAppliedStereotypeInstance().getClassifier()[0].
                    getName()=~/Port/)){
65                  if (att.type instanceof Class) {
66                      output[(att.getName())] = buildPropMap(att.
                            getType(),ALH.getValue(rto,att.getName()));
67                  }else{
68                      //println("Collecting a value...")
69                      currName = att.getName();
70                      //ALH.setValue(rto,currName,currVal+3.14159)//
                            test for changing values.
71                      output[(currName)]=ALH.getValue(rto,att.getName
                            ()) instanceof EnumerationLiteral ?
                            chkbuildres(ALH.getValue(rto,att.getName()))
                            : currVal ; // no loop on ALH methods without
                             the names from the map.
72                  }
73              }else{
74                  continue
75              }
76          }
77      }
78      return output.clone()
79  }
80
```

```
81 def getInstanceName = {aRto ->
82   return aRto.toString().split(':')[0].split().join('␣'); //name of
       instance
83 }
84
85  Gson gson = new GsonBuilder()
86      .serializeSpecialFloatingPointValues()
87      .create();
88 print(inVar)
89 print(inVar instanceof java.util.ArrayList)
90 print(inVar instanceof fUML.Semantics.Classes.Kernel.Object_)
91 print(inVar.class)
92 //print(inVar.size())
93 if (inVar instanceof java.util.ArrayList){
94     print("Parsing␣array␣list␣length␣${inVar.size()}")
95     obj = control.getTypes()[0]
96     //print("Current Object is: ${obj.getName()}")
97   itm_list = []
98   inVar.each{it ->
99       //print(it.toString())
100    thingmap = buildPropMap(control.getTypes()[0],it)
101    //print(thingmap)
102    itm_name = getInstanceName(it)
103    //thingmap["Feature_Kind"] = control.getTypes()[0].getName()
104    itm_list << thingmap
105  }
106  if (itm_list.size()==1){
107     output = gson.toJson(itm_list[0])
108  }else{
109     output = gson.toJson(itm_list)
110    }
111 }else if (inVar instanceof fUML.Semantics.Classes.Kernel.Object_){
112   thingmap = buildPropMap(control.getTypes()[0],inVar)
```

```
113     itm_name = getInstanceName(inVar)
114     //thingmap["Feature_Kind"] = control.getTypes()[0].getName()
115     output = gson.toJson(thingmap)
116 }else{
117     output = 'Error! Inappropriate use of ParseArgsToJson'
118 }
```

APPENDIX D
ARCHITECTING SPACECRAFT LICENSE

The following license is included in this spacecraft due to its association with the SysML model which accompanies Friedenthal and Oster[98][115].

D.1 License Text

ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The views and conclusions contained in the software and documentation are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the FreeBSD Project.

REFERENCES

[1] J. A. Estefan, "Survey of model-based systems engineering (MBSE) methodologies," INCOSE, Tech. Rep., 2008.

[2] E. C. Honour, "Systems Engineering Return on Investment," Ph.D. dissertation, Defense, systems institute, school of electrical, and information engineering, university of australia, 2013.

[3] *Incose systems engineering handbook*, International Council of Systems Engineers, 2015.

[4] J. Martin, *Systems Engineering Guidebook - A Process for Developing Systems and Products*. Boca Raton, FL: CRC, 1996.

[5] A. M. Madni and S. Purohit, "Economic Analysis of Model-Based Systems Engineering," *MDPI Systems*, vol. 12, no. 7, 2019.

[6] *Systems and software engineering - system life cycle processes*, ISO/IEC/IEEE 15288:2015, nternational Organization for Standardization (ISO)/International Electrotechnical Commission (IEC)/Institute of Electrical and Electronics Engineers (IEEE, 2015.

[7] D. P. Schrage, *Modern Systems Engineering — The Integration of Key Elements for System of Systems and Product Life-cycle Engineering*, Lecture Slides, 2006.

[8] *Systems and Software Engineering — Life Cycle Management — Part 2: Guide to the Application of ISO/IEC 15288 (System Life Cycle Processes)*, 2012, IEEE Computer Society.

[9] E. C. Honour and R. Valerdi, "Advancing an Ontology for Systems Engineering to Allow Consistent Measurement," in *Conference on Systems Engineering Research*, 2006.

[10] J. Fortune, R. Valerdi, B. M. Boehm, and F. S. Settles, "Estimating Systems Engineering Reuse," in *7th Annual Conference on Systems Engineering Research*, 2009.

[11] J. O. Grady, *System Requirements Analysis*, 1st ed. Burlington, MA: Elsevier Academic Press, 2006.

[12] B. S. Blanchard and W. J. Fabrycky, *Systems Engineering and Analysis*, 4th ed., W. Fabrycky and J. Mize, Eds. Pearson Prentice Hall, 2006.

[13] *Systems and software engineering — architecture description*, 42010:2011, International Standard, 2011.

[14] R. Rasmussen, "Principled System Architecture — prerequisite for resilience," in *Keck Institute for Space Studies — Workshop: Engineering Resilient Space Systems*, California Institute of Technology, Jul. 2012.

[15] J. O. Grady, *System Verification - Proving the Design Solution Satisfies the Requirements*. Elsevier Academic Press, 2016.

[16] A. W. Wymore, *Model-Based Systems Engineering*. CRC Press, 1993.

[17] ——, *A Mathematical Theory of Systems Engineering — The Elements*. New York: John Wiley and Sons, Inc., 1967.

[18] L. Delligatti, *SysML Distilled - A Brief Guide to the Systems Modeling Language*. Addison-Wesley Professional, 2013.

[19] W. L. Chapman, A. T. Bahill, and A. W. Wymore, *Engineering Modeling and Design*. CRC, 1992.

[20] P. Pearce and M. Hause, "ISO-15288, OOSEM, and Model-Based Submarine Design," in *SETE APCOSE*, 2012.

[21] M. D. Ingham, R. D. Rasmussen, M. B. Bennett, and A. C. Moncada, "Engineering Complex Embedded Systems with State Analysis and the Mission Data System," *Journal of Aerospace Computing, Information, and Communication*, vol. 2, pp. 507–536, 2005.

[22] J.-F. Castet, M. L. Rozek, M. D. Ingham, N. F. Rouquette, S. H. Chung, A. A. Kerzhner, K. M. Donahue, J. S. Jenkins, D. A. Wagner, D. L. Dvorak, and R. Karban, "Ontology and Modeling Patterns for State-Based Behavior Representation," in *AIAA Scitech*, American Institute of Aeronautics and Astronautics, Kissimmee, FL, 2015.

[23] M. E. Gooden, "Return on Investment for Complex Projects Utilizing Model Based Systems Engineering (MBSE)," in *NSBE Space SIG*, 2016.

[24] G. A. Hazelrigg, *Fundamentals of Decision-Making: For Engineering Design and Systems Engineering*. Self, 2012.

[25] R. Valerdi, "The Constructive Systems Engineering Cost Model (COSYSMO)," Ph.D. dissertation, University of Southern California, 2005.

[26] J. Brooks Frederick P., *The Mythical Man-Month (anniversary ed.)* 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc, 1995.

[27] R. Valerdi, G. J. Roedler, J. E. Rieff, M. J. Wheaton, and G. Wang, "Lessons Learned From Industrial Validation of COSYSMO," *INCOSE*, 2007.

[28] J. A. Lane, "Cost Model Extensions to Support Systems Engineering Cost Estimation for Complex Systems and Systems of Systems," in *7th Annual Conference on Systems Engineering Research 2009 (CSER 2009)*, 2009.

[29] R. Valerdi and H. L. Davidz, "Empirical Research in Systems Engineering: Challenges and Opportunities of a New Frontier," *Systems Engineering*, vol. 12, no. 2, pp. 169–181, 2009.

[30] J. Fortune and R. Valerdi, "A Framework for Reusing Systems Engineering Products," *Systems Engineering*, vol. 16, no. 3, pp. 304–312, 2013.

[31] B. Clark, S. Devnani-Chulani, and B. Boehm, "Calibrating the COCOMO II Post-Architecture model," in *Proceedings of the 20th International Conference on Software Engineering*, Kyoto, Japan: IEEE, 1998.

[32] R. Peak and J. A. Lane, "SysML Building Blocks for Cost Modeling: Towards Model-Based Affordability Analysis," in *INCOSE International Workshop*, 2014.

[33] J. Fortune, "Estimating Systems Engineering Reuse with the Constructive Systems Engineering Cost Model (COSYSMO 2.0)," Ph.D. dissertation, University of Southern California, 2009.

[34] R. Madachy and D. Jacques, "System Cost Modeling and SysML Integration in Model-Based Systems Engineering," in *INCOSE San Diego Chapter Meeting*, 2017.

[35] J. O. Grady, *System Requirements Analysis*, 2nd ed. Burlington, MA: Elsevier Academic Press, 2014.

[36] E. C. Honour, "Understanding the Value of Systems Engineering," in *14th Annual International Symposium Proceedings Managing Complexity and Change*, INCOSE, 2004.

[37] B. Papke, S. Pavalkis, and G. Wange, "Enabling Repeatable SE Cost Estimation with COSYSMO and MBSE," in *27th Annual INCOSE International Symposium (IS 2017)*, Adelaide, Australia: INCOSE, Jul. 2017.

[38] W. W. Hines, D. C. Montgomery, D. M. Goldsman, and C. M. Borror, *Probability and Statistics in Engineering*, 4th ed. Wiley, 2003.

[39] Measurement Working Group, *Systems Engineering Measurement Primer - A Basic Introduction to Measurement Concepts and Use for Systems Engineering*, INCOSE-TP-2010-005-02, INCOSE, 2010.

[40] D. H. Rhodes, R. Valerdi, and G. Roedler, "Systems Engineering Leading Indicators for Assessing Program and Technical Effectiveness," *Systems Engineering*, vol. 12, no. 1, pp. 21–35, 2009.

[41] G. Roedler, D. H. Rhodes, H. Schimmoller, and C. Jones, "Systems Engineering Leading Indicators Guide," INCOSE LAI PSM SEARI, Tech. Rep., 2010.

[42] T. Walworth, M. Yearworth, L. Shrieves, and H. Sillitto, "Estimating Project Performance through a System Dynamics Learning Model," *Systems Engineering*, vol. 19, no. 4, pp. 334 –350, 2016.

[43] M. W. Grenn, S. Sarkani, and T. Mazzuchi, "The Requirements Entropy Framework in Systems Engineering," *Systems Engineering*, vol. 17, no. 4, 2014.

[44] M. Pena and R. Valerdi, "Characterizing the Impact of Requirements Volatility on Systems Engineering Effort," *Systems Engineering*, vol. 18, no. 1, pp. 59–70, 2015.

[45] P. Micouin, "Toward a property based requirements theory: System requirements structured as a semilattice," *Systems Engineering*, vol. 11, no. 3, pp. 235–245, Apr. 2008.

[46] M. Jahangirian, T. Eldabi, A. Naseer, L. K. Stergioulas, and T. Young, "Simulation in manufacturing and business: A review," *European Journal of Operational Research*, vol. 203, pp. 1–13, 2010.

[47] J. M. Lyneis, "System dynamics for market forecasting and structural analysis," *System Dynamics Review*, vol. 16, no. 1, pp. 3–25, 2000.

[48] L. Rabelo, M. Helal, A. Jones, and H.-S. Min, "Enterprise simulation: A hybrid system approach," *International Journal of Computer Integrated Manufacturing*, vol. 18, no. 6, pp. 498–508, 2005.

[49] J. Venkateswaran, Y.-J. Son, and A. Jones, "Hierarchical production planning using a hybrid system dynamic-discrete event simulation architecture," in *Proceedings of the 2004 Winter Simulation Conference*, IEEE, Washington, D.C., USA, 2004.

[50] M. Broy, K. Havelund, and R. Kumar, "Towards a unified view of modeling and programming," in *ISoLA 2016, Part II, LNCS 9953*, T. Margaria and B. Steffen, Eds., Springer International Publishing AG, 2016, pp. 238–257.

[51] S. Breiner, E. Subrahmanian, and A. Jones, "Disciplinary convergence in systems engineering," in A. Madni, B. Boehm, R. Ghanem, D. Erwin, and M. Wheaton, Eds. Springer International Publishing AG, 2018, ch. Categorical Foundations for System Engineering, pp. 449–463.

[52] M. A. Mabrok and M. J. Ryan, "Category theory as a formal mathematical foundation for model-based systems engineering," *Appl. Math. Inf. Sci.*, vol. 11, no. 1, pp. 43–51, Jan. 2017.

[53] M. Broy, "A Logical Approach to Systems Engineering Artifacts: Semantic relationships and dependencies beyond traceability–from requirements to functional and architectural views," *Software & Systems Modeling*, vol. 17, no. 2, pp. 365–393, 2018.

[54] ——, "Theory and methodology of assumption/commitment based system interface specification and architectural contracts," *Formal Methods of System Design*, vol. 52, no. 1, pp. 33–87, 2018.

[55] M. Broy and K. Stølen, *Specification and Development of Interactive Systems: FOCUS on Streams, Interfaces, and Refinement*, D. Gries and F. B. Schneider, Eds., ser. Monographs in Computer Science. New York: Springer-Verlag, 2001.

[56] B. P. Zeigler, A. Muzy, and E. Kofman, *Theory of Modeling and Simulation — Discrete Event and Iterative System Computational Foundations*, 3rd ed. Elsevier Academic Press, 2018.

[57] A. Muzy, B. P. Zeigler, and F. Grammont, "Iterative Specification as a Modeling and Simulation Formalism for I/O General Systems," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2982–2993, 2018.

[58] A. Sudol and D. N. Mavris, "Framework for relability growth and rework projections for launch vehicles during testing," in *IEEE Aerospace Conference*, Big Sky, MT, SUSA, 2018.

[59] S. Chouali, M. Heisel, and J. Souquières, "Proving Component Interoperability with B Refinement," *Electronic Notes in Theoretical Computer Science*, vol. 160, pp. 157–172, Aug. 2006.

[60] I. Mouakher, A. Lanoix, and J. Souquières, "Component Adaptation: Specification and Verification," in *11th International Workshop on Component Oriented Programming - WCOP 2006*, LORIA - CNRS - Université Nancy 2, Nantes, France, Jul. 2006.

[61] K. A. Reilley, S. J. Edwards, R. S. Peak, and D. N. Mavris, "Methodologies for Modeling and Simulation in Model-Based Systems Engineering Tools," in *AIAA Space Forum*, Long Beach, CA, 2016.

[62] K. Reilley, S. Cimtalay, and D. N. Mavris, "Analysis-Centric Template Enabling Requirements Validation Through Engineering Models," in *CASI AERO*, Montreal, QC, CA, 2019.

[63] B. F. Cole and J. S. Jenkins, "Connecting requirements to architecture and analysis via model-based systems engineering," in *AIAA SciTech*, AIAA, 2015.

[64] B. F. Cole and K. Dinkel, "Multidisciplinary Model Transformation through Simplified Intermediate Representation," in *2016 IEEE Aerospace Conference*, California Institute of Technology, Big Sky, MT, USA: IEEE, Mar. 2016.

[65] T. Sprock and L. F. McGinnis, "Analysis of functional architectures for discrete event logistics systems (dels)," in *Procedia Computer Science 2015 Conference on Systems Engineering Research*, vol. 44, 2015, pp. 517–526.

[66] SAE, "Guide for development of civil aircraft and systems," Society of Automotive Engineers, Tech. Rep. ARP 4754A, 2010.

[67] R. G. Sargent, "Verification and Validation of Simulation Models," in *Proceedings of the 2010 Winter Simulation Conference*, B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, Eds., IEEE, 2010, pp. 166–183.

[68] R. S. Carson and P. V. Kohl, "New Opportunities for Architecture Measurement," in *INCOSE International Symposium*, 2013.

[69] R. S. Carson, "Differentiating System Architectures: Applying Architecture Measures," in *INCOSE International Symposium*, 2015.

[70] M. Jackson, M. Wilkerson, and J.-F. Castet, "Exposing hidden parts of the se process: Mbse patterns and tools for tracking and traceability," in *IEEE Aerospace Conference*, NASA Jet Propulsion Laboratory, California Institute of Technology, Big Sky, MT, USA: IEEE, 2016.

[71] D. N. Mavris, O. Bandte, and D. A. DeLaurentis, "Robust design simulation: A probabilistic approach to multidisciplinary design," *Journal of Aircraft*, vol. 36, no. 1, pp. 298–307, 1999.

[72] J. Sobieszczanski-Sobieski, "Advances in Structural Optimization," in J. Herskovits, Ed. Springer, Dordrecht, 1995, vol. 25, ch. Mutildisciplinary Design Optimization: An Emerging New Engineering Discipline, pp. 483–496.

[73] M. Balesdent, N. Berend, P. Depince, and A. Chriette, "A survey of multidisciplinary design optimization methods in launch vehicle design," *Structural Multidisciplinary Optimization*, vol. 45, pp. 619–642, 2012.

[74] D. J. Pate, J. Gray, and B. J. German, "A graph theoretic approach to problem formulation for multidisciplinary design analysis and optimization," *Structural and Multidisciplinary Optimization*, vol. 49, no. 5, pp. 743–760, 2014.

[75] I. v. Gent, R. Lombardi, G. La Rocca, and R. d'Ippolito, "A fully automated chain from mdao problem formulation to workflow execution," in *EUROGEN 2017*, Madrid, Spain, 2017.

[76] I. v. Gent, G. La Rocca, and L. L. M. Velhuis, "Composing mdao symphonies: Graph-based generation and manipulation of large multidisciplinary systems," in *AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, AIAA AVIATION Forum, Denver, CO, 2017.

[77] T. R. Browning, "Applying the design structure matrix to system decomposition and integration problems: A review and new directions," *IEEE Transactions on Engineering Management*, vol. 48, no. 3, pp. 292–306, 2001.

[78] A. B. Lambe and J. R. R. A. Martins, "Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes," *Structural Multidisciplinary Optimization*, vol. 46, pp. 273–284, 2012.

[79] S. Diagne, A. Coulibaly, and F. De Bertrand De Beuvron, "Complex product modeling based on a multi-solution extended conceptual design semantic matrix for behavioral performance assessment," *Computers in Industry*, vol. 75, pp. 101–115, 2016.

[80] C. D. Jilla and D. W. Miller, "Multi-Objective, Multidisciplinary Design Optimization Methodology for Distributed Satellite Systems," *Journal of Spacecraft and Rockets*, vol. 41, no. 1, pp. 39–50, Jan. 2004.

[81] G. S. Parnell, Ed., *Trade-Off Analytics: Creating and Exploring the System Tradespace*. Wiley, 2016.

[82] I. Ceh, M. Crepinsek, T. Kosar, and M. Mernik, "Ontology driven development of domain-specific languages," *Computer Science and Information Systems*, vol. 8, no. 2, pp. 317–342, 2011.

[83] M. Balesdent, L. Brevault, N. B. Price, S. Defoort, R. L. Riche, N.-H. Kim, R. T. Haftka, and N. Bérend, "Space engineering," in G. Fasano and J. Pintér, Eds., ser. Springer Optimization and Its Applications. Springer International Publishing, 2016, vol. 114, ch. Advanced Space Vehicle Design Taking into Account Multidisciplinary Couplings and Mixed Epistemic/Aleatory Uncertainties.

[84] S. J. Edwards, M. J. Diaz, D. N. Mavris, and D. J. Trent, "A Model-Based Framework for Synthesis of Space Transportation Architectures," in *AIAA SPACE Forum*, 2018 AIAA SPACE, Astronautics Forum, and Exposition, Sep. 2018.

[85] P. Geyer, "Multidisciplinary grammars supporting design optimization of buildings," *Research in Engineering Design*, vol. 18, no. 4, pp. 197–216, 2008.

[86] ——, "Component-oriented decomposition for multidisciplinary design optimization in building design," *Advanced Engineering Informatics*, vol. 23, pp. 12–31, 2009.

[87] D. Bohnke, A. Reichwein, and S. Rudolph, "Design language for airplane geometries using the unified modeling language," in *Proceedings of the ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, 2009.

[88] S. Rudolph, P. Arnold, M. Eheim, S. Hess, M. Motzer, M. Riestenpatt, J. Schmidt, and R. Weil, "Design languages for multi-disciplinary architectural synthesis and analysis of complex systems in the context of an aircraft cabin," in *CEAS Conference*, Similarity Mechanics Group, Institute for Statics and Dynamics of Aerospace Structures (ISD), University of Stuttgart, Germany, Toulouse, Nov. 2014.

[89] J. Gross and S. Rudolph, "Modeling graph-based satellite design languages," *Aerospace Science and Technology*, vol. 49, pp. 63–72, 2015.

[90] ——, "Rule-based spacecraft design space exploration and sensitivity analysis," *Aerospace Science and Technology*, vol. 59, pp. 162–171, 2016.

[91] J. R. Wertz, D. F. Everett, and J. J. Puschell, Eds., *Space Mission Engineering: The New SMAD (Space Technology Library, Vol. 28)*. Microcosm Press, 2011, ISBN: 978-1-881-883-15-9.

[92] G. J. Friedman and P. Phan, *Constraint Theory: Multidimensional Mathematical Model Management*. Springer Nature, 2017.

[93] A. A. Shah, "Combining mathematical programming and sysml for component sizing as applied to hydraulic systems," M.S. thesis, Georgia Institute of Technology, 2010.

[94] F. E. Cellier and S. Frabricius, *SystemDynamics Library*, Online, 2013.

[95] S. Vogel and P. Arnold, "Object-orientation in graph-based design grammars," Dec. 19, 2017. arXiv: `1712.07204v1 [cs.SE]`.

[96] C. Scherrer, *Fast and flexible probabilistic programming with soss.jl*, webpage, Sep. 2019.

[97] C. Carroll, *A tour of probabilistic programming language apis*, webpage, Jul. 2019.

[98] S. Friedenthal and C. Oster, *Architecting Spacecraft with SysML - A Model-Based Systems Engineering Approach*. 2017, ISBN: 1544288069.

[99] T. Kulkarni, K. DeBruin, A. Nelessen, K. A. Reilley, R. Peak, S. J. Edwards, and D. N. Mavris, "A model based systems engineering approach towards developing a rapid analysis and trades environment," in *AIAA SPACE 2016*, 2016.

[100] R. L. Gilbertson, "Applying the Cynefin Sense-Awareness Framework to Develop a Systems Engineering Method Diagnostic Assessment Model SEMDAM," Ph.D. dissertation, George Washington University, 2018.

[101] A. Morkevicius, A. Aleksandraviciene, A. Armonas, and G. Fanmuy, "Towards a common systems engineering methodology to cover a complete system development process," in *INCOSE International Symposium*, 2020.

[102] S. Kleiner and C. Kramer, "Model based design with systems engineering based on RFLP using V6," in *Smart product engineering*, M. Abramovici and R. Stark, Eds., ser. Lecture notes in production engineering, Berlin: Springer, 2013.

[103] J.-L. Voirin, *Model-based system and architecture engineering with the arcadia method*. ISTE Press, Elsevier, 2017.

[104] I. Douven, "Abduction," in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed., Summer 2017, Metaphysics Research Lab, Stanford University, 2017.

[105] T. McDermott, "Modeling and Simulation in the Systems Engineering Life Cycle," in M. L. Loper, Ed., ser. Simulation Foundations, Methods and Applications. Georgia Tech Research Institute, Atlanta, GA: Springer-Verlag, May 2015, ch. Systems Thinking, pp. 273–297.

[106] G. E. Mobus and M. C. Kalton, *Principles of Systems Science*. Springer New York, 2015.

[107] J. Morecroft, *Strategic Modelling and Business Dynamics: A Feedback Systems Approach*. Chichester, England: John Wiley & Sons Ltd, 2007.

[108] Z. Jelinski and P. Moranda, "Software reliability research," *Statistical Computer Performance Evaluation*, pp. 465 –484, 1972.

[109] Duke. (Apr. 2019). "Python Ordinary Differential Equations/Examples." eprint: `https://pundit.pratt.duke.edu/wiki/Python:Ordinary_Differential_Equations/Examples#Preamble`.

[110] B. Cole, "Integration of SysML models with M&S tools for behavior modeling and simulation," in *Driving Industry to Model-Based Systems Engineering*, NASA, Troy, Michigan: Jet Propulsion Laboratory, Apr. 2016.

[111] J. M. Lyneis and D. N. Ford, "System dynamics applied to project management: A survey, assessment, and directions for future research," *System Dynamics Review*, vol. 23, no. 2/3, pp. 157–189, Jun. 2007.

[112] D. A. Wagner, M. B. Bennet, R. Karban, N. Rouquette, S. Jenkins, and M. Ingham, "An Ontology for State Analysis: Formalizing the Mapping to SysML," in *IEEE Aerospace Conference*, NASA, Big Sky, MT, USA: IEEE, Mar. 2012.

[113] M. Kordon, S. Wall, H. Stone, W. Blume, J. Skiper, M. Ingham, J. Neelon, J. Chase, R. Baalke, D. Hanks, J. Salcedo, B. Solish, M. Postma, and R. Machuzak, "Model-Based Engineering Design Pilots at JPL," in *IEEE Aerospace Conference*, NASA, Big Sky, MT, USA: IEEE, Mar. 2007.

[114] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML*, Third. Morgan Kaufmann, 2014, ISBN: 9780128002025.

[115] actiontofu, *Architecting-spacecraft*, GitHub Repository, Dec. 2017.

[116] M. Bott and B. Mesmer, "Determination of Function-Behavior-Structure Model Transition Probabilities from Real-World Data," in *AIAA SciTech Forum*, AIAA, 2019.

[117] J. V. Iacobucci, "Rapid architecture alternative modeling (raam)," Ph.D. dissertation, School of Aerospace Engineering, Georgia Institute of Technology, 2012.

[118] H. Abelson, G. J. Sussman, and J. Sussman, *Structure and Interpretation of Computer Programs*. The MIT Press, 1996.

[119] M. Odersky, L. Spoon, and B. Venners, *Programming in Scala*. Artima, 2019.

[120] N. Rouquette, *IMCE Dynamic Scripts plugin for MagicDraw 18*, Apr. 2020.

[121] A. A. Kerzhner, "Using logic-based approaches to explore system architectures for systems engineering," Ph.D. dissertation, Georgia Institute of Technology, 2012.

[122] J. L. Sharma, "STASE: Set Theory-Influenced Architecture Space Exploration," Ph.D. dissertation, Georgia Institute of Technology, 2014.

[123] I. E. Grossmann, "Review of nonlinear mixed-integer and disjunctive programming techniques," *Optimization and Engineering*, vol. 3, pp. 227–252, May 2002.

[124] A. Nelessen, "A model-based framework for exhaustive exploration of spacecraft design alternatives," research rep., Georgia Institute of Technology, 2015.

[125] J. Uotila, M. Maula, T. Keil, and S. A. Zahra, "Exploration, exploitation, and financial performance: Analysis of S&P 500 corporations," *Strategic Management Journal*, vol. 30, pp. 221–231, 2009.

[126] J. G. March, "Exploration and exploitation in organizational learning," *Organization Science*, vol. 2, no. 1, pp. 71–87, Feb. 2001.

[127] R. Wang, A. B. Nellippallil, G. Wang, Y. Yan, J. K. Allen, and F. Mistree, "Systematic design space exploration using a template-based ontological method," *Advanced Engineering Informatics*, vol. 36, pp. 163–177, 2018.

[128] J. Gross and S. Rudolph, "Dependency analysis in complex system design using the firesat example," in *INCOSE International Symposium*, 2014.

[129] R. S. Peak, R. M. Burkhart, S. A. Friedenthal, M. W. Wilson, M. Bajaj, and I. Kim, "Simulation-Based Design Using SysML - Part 2: Celebrating Diversity by Example," *INCOSE International Symposium*, vol. 17, no. 1, pp. 1516–1535, 2007.

[130] M. Bajaj, D. Zwemer, R. Peak, A. Phung, A. Scott, and M. Wilson, "Satellites to supply chains, energy to finance — SLIM for model-based systems engineering part 2: Applications of SLIM," in *INCOSE International Symposium*, vol. 21, 2011, pp. 410–431.

[131] G. T. Panorama, *Panorama plugin for magicdraw*.

[132] R. S. Peak, C. J. J. Paredis, L. F. McGinnis, S. A. Friedenthal, R. M. Burkhart, and M. Bajaj, "INCOSE model-based systems engineering (MBSE) challenge: Modeling & simulation interoperability (MSI) team status update - with applications to mechatronics, other cyber-physical systems, and beyond," in *INCOSE International Workshop*, Georgia Tech, Mesa, AZ, Jun. 2010.

[133] F. J. Dietrich, "Space Mission Analysis and Design," in W. J. Larson and J. R. Wertz, Eds., Third. Microcosm Press, ch. Communications Architecture, pp. 533–586.

[134] S. K. Rallabhandi and D. N. Mavris, "Sonic boom minimization using inverse design and probabilistic acoustic propagation," *Journal of Aircraft*, vol. 43, no. 6, pp. 1815–1828, Nov. 2006.

[135] *NASA Schedule Management Handbook*, Online, Mar. 2011.

[136] N. Hutchison, H. Y. S. Tao, W. Miller, D. Verma, and G. Vesonder, "Framework for mission engineering competencies," in *INCOSE International Symposium*, 2018.

[137] N. A. C. Hutchison, D. Verma, P. Burke, H. Y. S. Tao, R. Giffin, Z. Yu, D. Makwana, X. Chen, and Y. Xiao, "Wrt-1004: Helix," Systems Engineering Research Center, Tech. Rep. SERC-2020-TR-007, 2020.

[138] R. Beasley, D. Gelosh, M. Heisey, I. Presland, and L. Zipes, "INCOSE Systems Engineering Competency Framework," INCOSE, Tech. Rep. INCOSE-TP-2019-002-01.0, 2018.

[139] C. Whitcomb, J. Delgado, R. Khan, J. Alexander, C. White, D. Grambow, and P. Walter, "The Department of the Navy Systems Engineering Career Competency Model," in *Proceedings of the Twelfth Annual Acquisition Research Symposium*, 2015.

[140] K. Pepe, N. Hutchison, M. Blackburn, D. Verma, J. Wade, R. Peak, A. Baker, C. Whitcomb, and R. Khan, "Preparing the acquisition workforce: A digital engineering competency framework," in *INCOSE International Symposium*, 2020.

[141] J. Holt and S. Perry, *A Pragmatic Guide to Competency*. British Informatics Society Limited, 2011, ISBN: 9781906124700.